Investigación y desarrollo de una máquina virtual que permita la ejecución de agentes inteligentes en JavaScript. (JSWARS).

Luis Sebastián Huerta

Trabajo Fin de Estudios para
el grado de Ingeniería Informática

en

Ciencia de la Computación e Inteligencia Artificial
Universidad Carlos III Madrid

Tutores:
Prof. Carlos Linares López

Febrero 2016

Agradecimientos

A mis padres, por darme su apoyo incondicional y todas las ayudas necesarias para terminar este proyecto.

A mi hermano, por enseñarme las virtudes de LaTeXy ayudarme cuando ha sido necesario.

A mi tío Santiago, por su incesable insistencia y apoyo, por su frase favorita de Alien aplicada al proyecto 'La tripulación es sacrificable'. A mi compañero y amigo Marcos, que me ha acompañado durante esta senda en la que hemos trabajado muy duro juntos y nos hemos ayudado mutuamente a seguir adelante cuando las fuerzas flaqueaban.

A Juancar, por hacer de conejillo de indias en la aplicación y sus buenos consejos.

A mis compañeros de trabajo, por que han sido los mejores que he tenido nunca y me han apoyado hasta el final.

En especial a todos mis amigos que habéis estado apoyándome tanto todo este tiempo, Jorge, Carmen, Laura, Lucía y Oliva.

Abstract

The main goal of this project is to create a game engine for real time strategy games, designed for programmers to develop AI agents that will play the game. The engine may control all the elements of the game and it's iterations. As well as manage the code uploaded by the users and execute it in virtual machines. It was originally thought for academic purposes, for self-taught people or for using it in universities.

To conclude, it will be implemented a final AI agent for playing in the platform, which will be tested against all other agents. A prolonged research about the results of this matches will be made.v

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

La mayor motivación para realizar este proyecto ha sido cursar la especialidad de Computación, en todas asignaturas de esta rama nos han enseñado mucho, pero aún así es una pequeña parte de todo lo que este campo en expansión puede ofrecer. Ese es el motivo por el cual decidí realizar mi TFG(Trabajo Fin de Grado) en un tema relacionado con la IA.

Desde el punto de vista del alumno siempre me han gustado los retos de competiciones de Inteligencia Artificial, una buena manera de probar los conocimientos adquiridos en la materia a la vez que se puede observar de manera práctica su efectividad en un entorno controlado. El poder observar los resultados a corto plazo me fomentó a investigar más sobre la materia y probar de manera práctica nuevas técnicas de Inteligencia Artificial.

A partir de todas estas inquietudes surge la idea de crear una competición de Inteligencia Artificial propia sobre un juego de estrategia donde cada equipo estaría controlado por una Inteligencia Artificial creada por diferentes usuarios. Dado que el proyecto era grande en escala, me puse en contacto con un compañero de la universidad, Marcos Pérez Ferro, para que se encargase de toda la plataforma web donde gestionar la competición(véase [1]) y con nuestro tutor Carlos Lopez Linares que nos avalase ambos proyectos.

1.2. Área de la aplicación

Desde hace muchos años se utiliza la inteligencia artificial en muchos y distintos ámbitos, en el de la medicina por ejemplo se utiliza para realizar pronósticos y encontrar patrones en los análisis médicos, de manera que sea más fácil detectar enfermedades a tiempo a un coste menor.

Otro de los sectores que se aprovechan de los beneficios de las técnicas de IA es el de la publicidad, de esta manera viendo la información de los usuarios en internet y sus comportamientos se es capaz de categorizar y mostrar publicidad dirigida a usuarios cuyos intereses coinciden con el producto publicitado aumentando la tasa de conversión.

Si bien los videojuegos, es uno de los sectores cuya presencia de la inteligencia artificial siempre ha sido sumamente importante, no ha sido hasta el avance y popularización de computadoras potentes (ya sean ordenadores o consolas) cuando se ha podido aprovechar al máximo las técnicas que ofrecen, creando de esta forma un reto mayor para los jugadores, donde podían encontrar enemigos que no siempre vencieses, pero si que supusiesen un reto al poder desarrollar sus propias estrategias y aprender del jugador, haciendo que este tuviera que escaparse de su comodidad y buscar nuevas alternativas para vencer a la IA. El uso de inteligencias artificiales en videojuegos, ha derivado en enemigos más creíbles, humanizándolos en lugar de creando retos imposibles con inteligencias que vencen siempre.

La nueva visión de una IA cuyo objetivo sea parecer más humana supuso un nuevo reto con una complejidad muy alta que sigue suponiendo un reto incluso para los más grandes de la industria del videojuego. Ya no se trata de hacer una IA que sea la mejor, se trata de buscar el equilibrio para que el jugador sienta que tiene opciones de ganar pero a la vez esto le suponga un reto.

Capítulo 2

Estado de la cuestión

En esta sección se va a ofrecer una visión global sobre la Inteligencia Artificial y la creación de Agentes inteligentes, así como las plataformas existentes de competición en su ámbito.

La inteligencia artificial está en auge en múltiples aplicaciones y campos, usada sobre todo en medicina para hacer diagnósticos, en publicidad dirigida y uso de big data, tiene un gran uso en robótica, vehículos autónomos no tripulados y un uso muy extendido en videojuegos donde se intenta conseguir un comportamiento lo más parecido al humano para los enemigos y aliados y que la jugabilidad esté equilibrada.

En el ámbito académico han surgido desde hace algunos años competiciones online de inteligencia artificial aplicada a varios problemas, entre ellos el más común es el de los juegos, al ser un ámbito acotado y controlado se puede estudiar y medir la eficiencia de los algoritmos creados. Se va a realizar una retrospectiva de las competiciones más destacadas de los últimos años.

2.1. Google AI Challenge

La AIChallenge [2] tiene su origen en la universidad de Waterloo Computer Science Club en 2009, primeramente de forma privada y exclusiva para los alumnos de dicha universidad, y al año siguiente gracias al patrocinio de Google pasó a convertirse en una competición global abierta a todo el público.

Cada participante debía crear un agente de inteligencia artificial que fuese capaz de luchar contra un oponente o problema, y posteriormente subir su código a un servidor para competir.

Mediante un algoritmo de ranking TrueSkill [3] los agentes creados por los alumnos eran emparejados y competían en modo de torneo, pudiendo ver los resultados de forma online a medida que avanzaba la competición.

Desde el comienzo de esta competicion de periodicidad anual, los temas que se escogieron para competir fueron los siguientes:

- **Rock Paper Scissors - Otoño 2009:** En su primera edición tan sólo se desarrolló la competición de forma privada para los alumnos de University of Waterloo Computer Science Club. El tema

trataba sobre el conocido juego 'Piedra, papel o tijera' con una mecánica muy simple en la que cada turno los dos oponentes deben escoger una opción y siguiendo las reglas del juego (tijeras ganan a Papel, Papel gana a piedra y piedra gana a tijeras) se debe adivinar la elección del rival, para ello los agentes debían aprender de los resultados históricos de la partida para lograr una mejor puntuación que la que se consigue de forma aleatoria (1/3 de posibilidades de ganar, 1/3 de posibilidades de perder y 1/3 de posibilidades de empatar).

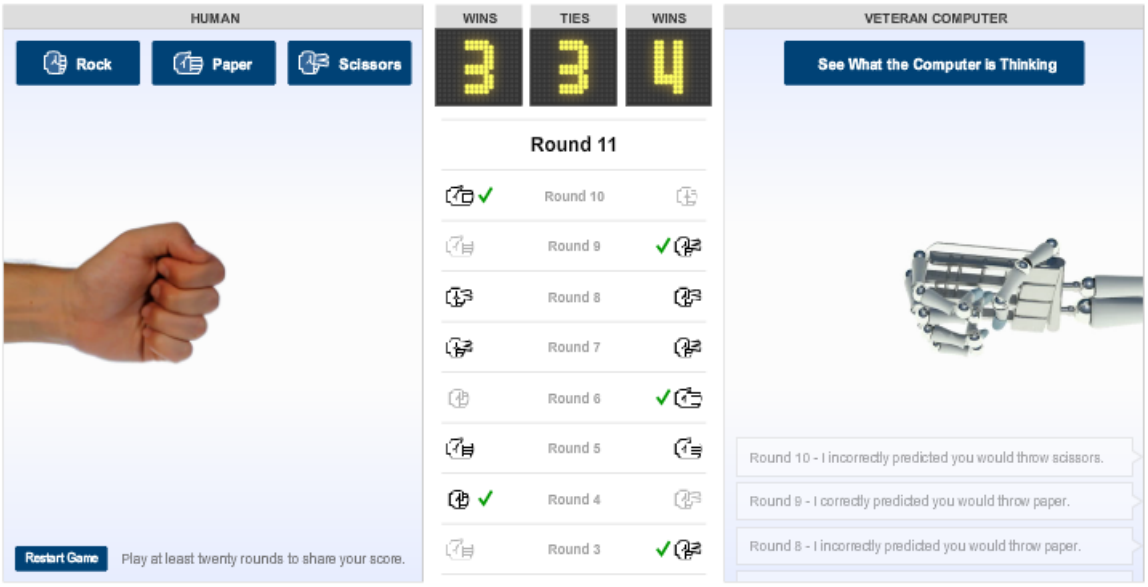

Figura 2.1: Partida de Piedra, Papel Tijera contra IA

- **Tron - Invierno 2010:** En la segunda edición bajo el patrocinio de Google la competición se extendió de forma internacional al resto alumnos y al público en general interesado. En este caso la mecánica es mas compleja, basado en la película de tron, dos jugadores en un mismo escenario se desplazan por una pantalla dividida en celdas dejando una estela por donde pasan, el primer jugador que colisione con el escenario o con una de las dos estelas pierde.

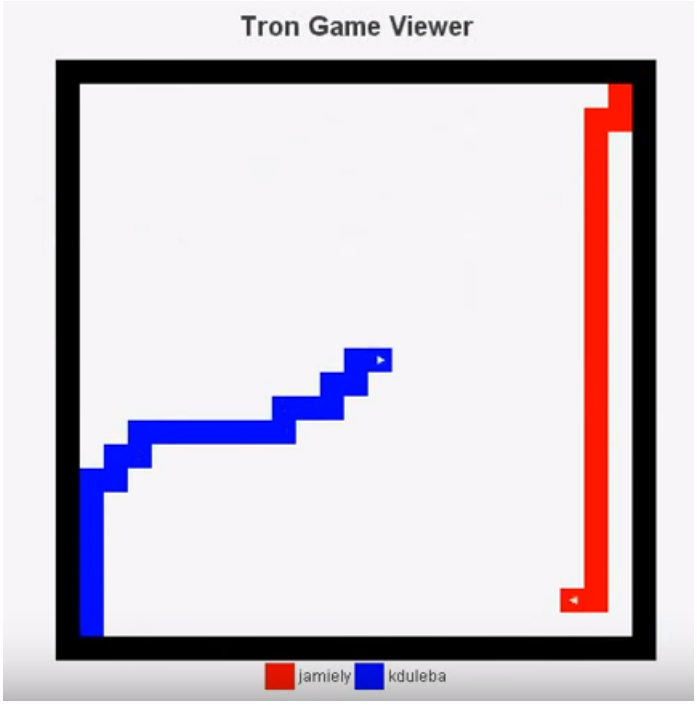

Figura 2.2: Partida de Tron, AIChallenge

- **Planet Wars - Otoño 2010:** Al año siguiente la competición se realizó sobre un nuevo juego con mecánica diferente. En un espacio de dos dimensiones con planetas de diferentes tamaños y ubicaciones compiten dos jugadores, los planetas se pueden colonizar mandando unidades y venciendo a los habitantes, iniciando la partida con todos los planetas neutrales salvo dos en los que se sitúa cada uno de los dos jugadores.

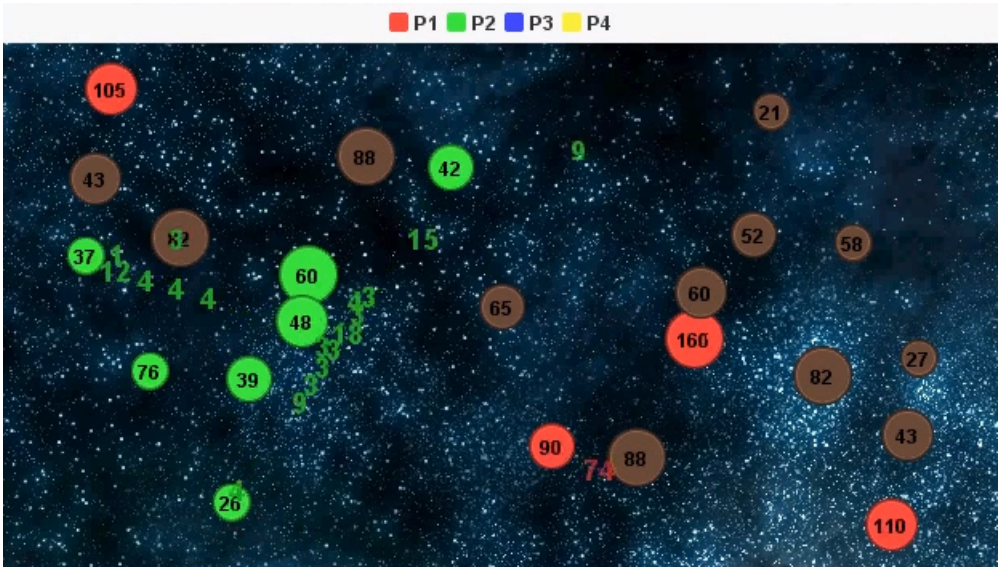

Figura 2.3: Partida de PlanetWars, AIChallenge

- **Ants - Otoño 2011:** La última competición realizada por AIChallenge [2] fue la del juego Ants, en este caso las partidas admitían 7 jugadores simultáneos en un mapa bidimensional. Separados por

celdas, cada equipo dispone de dos hormigueros situados en diferentes posiciones del mapa. Sus respectivos hormigueros aparecen hormigas de cada equipo y deben eliminar al resto a la vez que destruyen sus hormigueros para evitar que produzcan más.

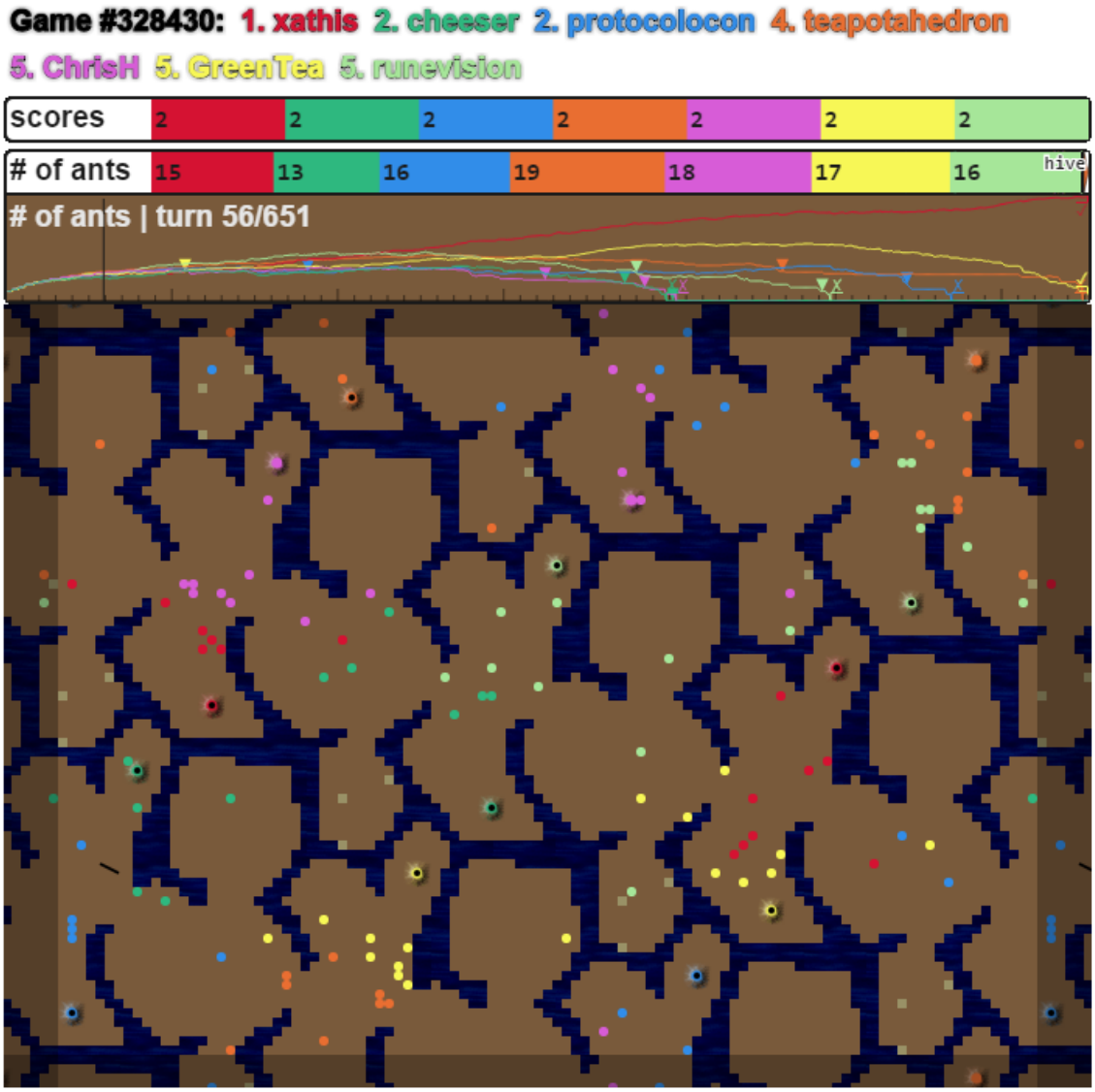

Figura 2.4: Partida de Ants, AIChallenge

2.2. AIBirds

Otra de las competiciones más conocidas de los últimos años es la AIBirds [4], basada en el popular juego conocido cómo Angry Birds, que parte con la premisa de conseguir un jugador de inteligencia artificial que consiga superar la mejor puntuación conseguida por los mejores jugadores Humanos.

Es un problema muy altamente complejo, que requiere que los agentes sean capaces de predecir el resultado de una acción en los objetos físicos del juego, sin tener conocimiento del mundo ni de las reacciones físicas de los objetos y sus comportamientos con colisiones.

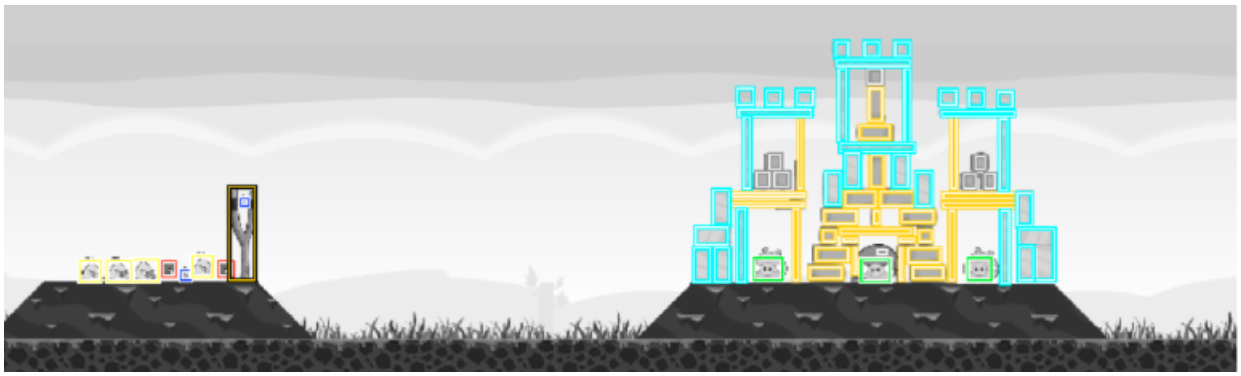

Figura 2.5: Representación gráfica Aibirds

2.3. General Video Game AI Competition

La competición GVG-AI [5], que es un proyecto del grupo de lógica de la universidad de Stanford en California. Explora el problema de enfrentar agentes de inteligencia artificial para que sean capaces de jugar problemas de videojuegos de manera general.

Los agentes creados para esta plataforma se enfrentarán a una amplia variedad de juegos, con distintas mecánicas y objetivos. Estos deben ser capaces de desenvolverse en todo ámbito de problemas de una manera satisfactoria.

Figura 2.6: Juego 1 - gvgai

Figura 2.7: Juego 2 - gvgai

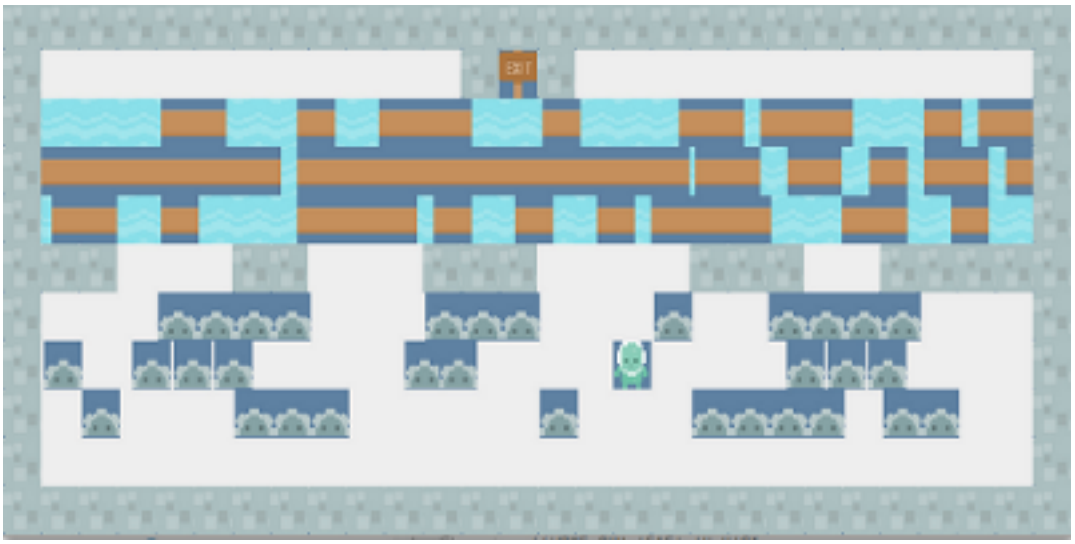

Figura 2.8: Juego 3 - gvgai

2.4. Mario AI Championship

Desarrollado en 2012, la competición Mario AI Championship [6] consiste en crear un agente que sea capaz de completar todos los niveles del popular juego Mario de manera autónoma. Para ello se provee al agente de todos los datos de su entorno: escenario, posición de los enemigos, posición de las monedas, desplazamiento lateral del nivel, etcétera.

A partir de estos datos, se puede evaluar el problema y crear una heurística para generar una inteligencia artificial que sea capaz de tomar decisiones y aprender a lo largo de varias iteraciones del juego y partidas.

Uno de los agentes creados por uno de los participantes en la competición es el siguiente mostrado en la figura 2.4. Su creador utiliza un algoritmo de path-finding A* para decidir las próximas acciones.

Figura 2.9: Partida de MarioAI

Otro participante decide hacer una implementación original que responde de manera conversacional, no sólo responde a órdenes como 'coge esa moneda' o 'mata a ese enemigo', sino que además es capaz de realizar deducciones lógicas para poder responder a preguntas como "¿Qué ocurre si saltas sobre ese enemigo? véase figura 2.4.

Figura 2.10: Partida de MarioAI

2.5. The AI Sandbox

Por último queda mencionar la competición que más nos ha inspirado para realizar este trabajo, AISandbox [7]. Consiste en un juego táctico al estilo de un Shooter, donde dos equipos compiten para capturar la bandera enemiga. El juego se caracteriza por seguir las mismas reglas que los juegos contemporáneos, si muere una unidad reaparece a los pocos segundos en su zona de inicio, las unidades

deben jugar en equipo para lograr su objetivo, además que hay un campo de visión y según los obstáculos del mapa pueden cambiar las estrategias y tácticas para lograr ganar. Aunque el motor de físicas interno del juego se comporta como si fuese un mapa bidimensional(ver figura 2.5), dada la gran repercusión que ha tenido esta competición, se han creado varios motores de renderizado de partidas que la representan en tres dimensiones.

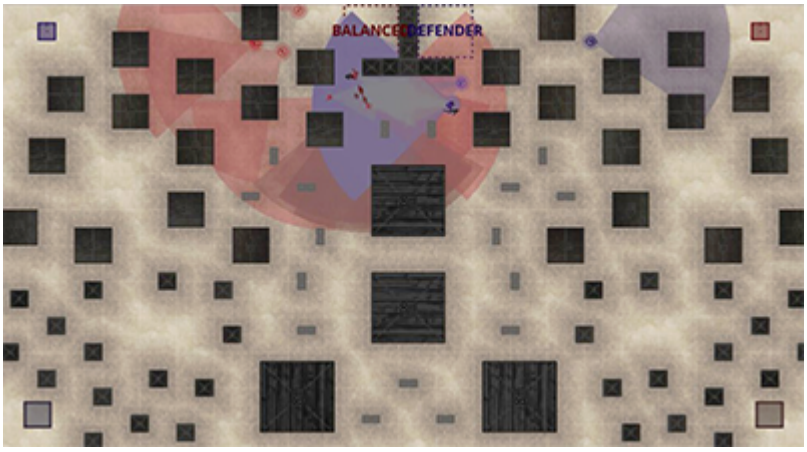

Figura 2.11: Partida de AISandbox, representación 2D

La mecánica es simple, se dispone de un mapa donde cada equipo esta situado en un lateral, dentro del mapa se colocan diversos obstáculos de diferentes tamaños a través de los cuales no se pueden ni atravesar caminando ni disparar a través de ellos. Las unidades pueden moverse en comandos o por separado, cada unidad tiene un campo de visión limitado por un ángulo y una distancia, teniendo visión directa de un enemigo las unidades pueden atacarlo. La precisión del tiro va limitada a distancia y otros factores, por lo que si un comando tiene más unidades que el comando enemigo con el se encuentra tendrá mas posibilidades de ganar ese combate, lo que hace que el factor táctico y de estrategia sea muy importante para lograr la victoria. En la siguiente figura 2.5 se puede ver una representación de la partida en tres dimensiones.

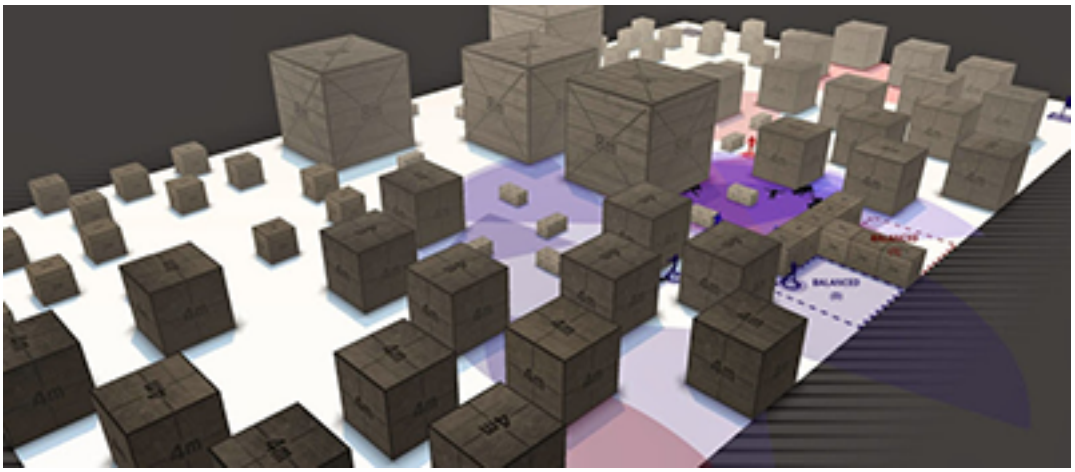

Figura 2.12: Partida de AISandbox, representación 3D

Capítulo 3

Objetivos

El objetivo es crear un motor de juego que permita la creación y evaluación de agentes en Javascript, que estarán a cargo de un conjunto de unidades. Estas unidades pueden moverse y disparar con el fin de derrotar a todas las unidades enemigas. El motor de juego estará a cargo de ejecutar las partidas, así como controlar el comportamiento de las mismas, movimientos, disparos, físicas de colisión, etc.

El proyecto está dividido en dos partes, cada una de ellas realizada por un integrante del grupo. En la parte que vamos a desarrollar en este documento, los objetivos principales son:

- Implementación de una de **máquina virtual sobre NodeJS** que permita ejecutar los distintos agentes de los usuarios sobre un entorno controlado y protegido. El ciclo completo de ejecución de la batalla se basa en tres conceptos básicos: contexto persistente (variables compartidas en la máquina virtual entre iteraciones de la misma batalla), método init (llamado por la máquina virtual una única vez para preparar el contexto persistente) y método tick (llamado por la máquina virtual en cada iteración).
- Preparación de una **documentación** que permita a los usuarios crear sus propios agentes. En esta documentación se expondrá la API que está disponible en el momento de ejecución del agente.
- Identificación del **tipo de problema**, obtener una visión global del problema y estudiar cual sería la mejor forma de abordarlo.
- **Creación de agentes** siguiendo la API definida en la plataforma. Se crearán **agentes** usando algoritmos de inteligencia artificial.
- **Enfrentar** en la plataforma **los agentes** creados unos contra otros, guardando la ejecución y resultados de todas las partidas.
- **Analizar** los resultados de las batallas entre los agentes creados, y evaluar de forma objetiva cada uno de los resultados obtenidos.

- Realizar **un estudio sobre todos los agentes subidos** por otros usuarios a la plataforma, comparando sus resultados en la competición y analizando los métodos utilizados.
- **Utilizar los conocimientos adquiridos** tras evaluar todas las etapas anteriores **para crear un último agente** o mejorar uno de los anteriores con el propósito de obtener mejores resultados, usando para entrenar también los datos obtenidos de todas las partidas que se hayan jugado en la plataforma.

Capítulo 4

Desarrollo del Motor de juego

En este apartado se van a describir todos los pasos relativos al desarrollo del software del motor de juego que se ejecutará en el servidor. A partir del diseño hasta su implementación y posterior puesta en producción.

Primeramente se analizará de forma detallada las necesidades del sistema y posteriormente se expondrán los requisitos del desarrollo del software de una manera detallada.

Una vez definidas las necesidades básicas de nuestro motor de juego se procederá a buscar una arquitectura que se adecue al proyecto. Mediante este análisis se estudiarán las arquitecturas actuales y mediante una evaluación en función a los casos de uso se escogerá la que mejor se adapte al proyecto.

4.1. Objetos del sistema

En esta sección se va a realizar un análisis de los objetos dependientes del sistema y de su relación entre ellos, explicando su estructura, dependencias y las principales características de cada uno. En la figura 4.1 se muestra un posible ejemplo de jerarquía entre objetos, a partir del cual se basará el diseño del sistema.

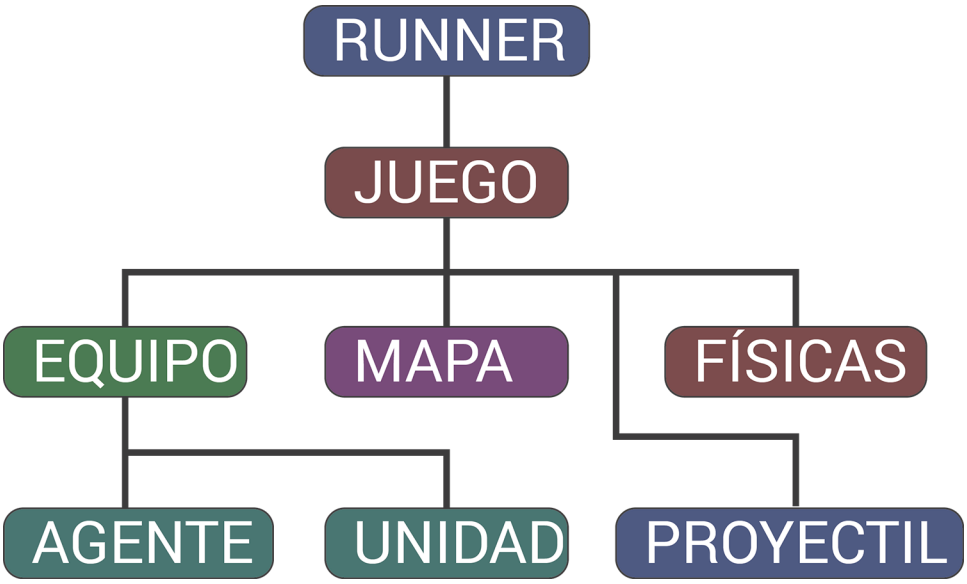

Figura 4.1: Diagrama de Objetos del motor del juego

4.1.1. Juego

El objeto juego será el encargado de gestionar la partida, conteniendo así al resto de objetos y estableciendo una relación entre los mismos. Se encargará de instanciar, procesar e inicializar el mapa del juego donde se desarrollará la batalla. También se encargará de instanciar los correspondientes equipos y los agentes asociados que darán las ordenes a cada una de las Unidades instanciadas de su equipo. Por otro lado todos los proyectiles en juego estarán contenidos en el objeto Juego.

El objeto juego cumplirá una función importante a la hora de gestionar el flujo de la batalla. Gestionando cada iteración del juego, enviando el estado del juego a los agentes, y recibiendo las órdenes de cada uno de los agentes para aplicarlas a las unidades de sus correspondientes Equipos. Además se encargará de comprobar el final de la partida, ya sea por eliminación de uno de los dos equipos o por finalización del tiempo límite.

4.1.2. Mapa

El objeto mapa se encargará de procesar los mapas de las diferentes partidas, consistirá en una matriz de colisiones indicando las zonas sólidas del mapeado, mediante sus atributos se determinará el alto, el ancho del mismo y la escala, que permitirá hacer mapas con mayor precisión a la hora de poner obstáculos, permitiendo así una mayor escalabilidad. Véase la figura 4.1.2

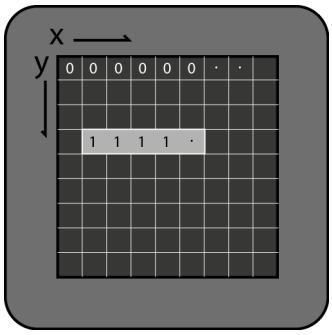

Figura 4.2: Array de colisiones Mapa

El mapa se podrá generar mediante un editor de mapas OpenSource Tiled [8] y podrá ser importado en la plataforma. La figura 4.1.2 muestra una representación gráfica de un mapa de ejemplo.

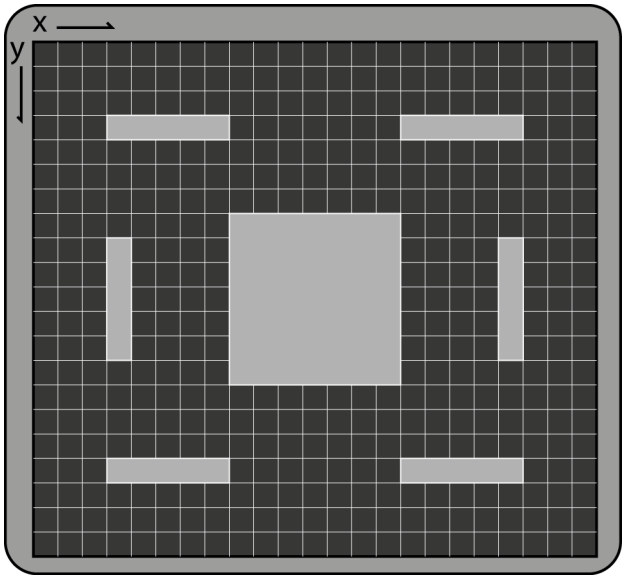

Figura 4.3: Representación gráfica del mapa

Adicionalmente, el objeto mapa pondrá a disposición del resto de objetos una función para comprobar las colisiones con el mismo, además de proporcionar a los agentes una función de pathfinding utilizando A* indicándolo así en la API para que los usuarios de la plataforma puedan aprovechar dicha función.

4.1.3. Equipo

Cada uno de los equipos se á compuesto por un agente asociado que lo controla, un array de unidades, el color del equipo, la vida total (que será una suma de la vida de todas sus unidades) y un indicador de si el equipo está vivo o no, que se calculará en función de la vida restante de las unidades que lo componen.

4.1.4. Unidad

Las unidades de los equipos estan compuestas de los siguientes atributos:

- **Identificador de equipo:** variable con el valor del identificador del equipo al que corresponde la unidad

- **Posición:** indica la posición de la Unidad en el mapa, la posición es relativa a la esquina superior izquierda, y la celda dentro del mapa que ocupa la unidad es el resultado de multiplicar la posición de la unidad por la escala del mapa en sus dos componentes, x e y, además de aplicarles una función suelo para poder acceder a su celda en el array de colisiones del Mapa.
- **Radio:** indica el tamaño de la unidad, la unidad se ve afectada por su posición y su radio para las colisiones con el entorno del Mapa y para calcular los impactos de los proyectiles de unidades enemigas.
- **Velocidad:** señala la velocidad máxima a la que se puede desplazar la unidad por cada iteración del Juego. Si la distancia a la posición indicada por el agente para mover la unidad es superior a la velocidad, se desplazará en esa dirección la magnitud indicada por esta variable, en caso contrario de indicar una posición más cercana a dicho parámetro, la unidad llegará a su destino sin pasarse.
- **Vida:** referencia la vida restante de la unidad, cuando es golpeada por un proyectil enemigo su vida irá decrementándose hasta llegar al valor cero, indicando así que la unidad está muerta.
- **Armadura:** este valor indica la resistencia a los daños disminuyendo el impacto de los mismos en un porcentaje.
- **Daño:** indica la cantidad de daño que producirán los ataques de la unidad, en este caso serían los proyectiles que la unidad lance, pero está pensado para que sea escalable y se contemplen otros tipos de ataques o unidades. Por ejemplo unidades cuerpo a cuerpo, también se ha planteado implementar powerups que modifiquen esta variable durante la partida dado que el sistema esta preparado para ello. En un primer momento y con la intención de mantener un sistema robusto y sencillo, se ha decidido no implementarlo en la primera versión.
- **Cadencia de tiro:** indica la velocidad de disparo que tiene la unidad, el número de iteraciones del juego que tienen que pasar entre tiro y tiro. Cuanto menor sea este número mas rápido disparará la unidad.
- **Alcance:** señala la distancia a la que pueden llegar los proyectiles de la unidad, actualmente las unidades por defecto tienen más alcance que el tamaño del mapa y por lo tanto viajarán hasta colisionar con otra unidad o con el Mapa. Para futuras implementaciones o tipos de unidades esta variable se puede modificar y es escalable con el resto del sistema de juego.
- **Tipo de unidad:** actualmente tan sólo hay un tipo de unidad (Unidad de ataque a distancia), pero se plantea añadir distintos tipos de unidades más adelante.

Todas las variables de la Unidad son dinámicas y el sistema está preparado para que puedan modificar sus valores durante la partida, además de existir la posibilidad de crear distintos tipos de unidades que hereden de este objeto con distintos atributos y/o tipos de ataque.

Adicionalmente las unidades disponen de las funciones necesarias para recibir las órdenes de los agentes de movimiento y de ataque, de tal forma que en cada iteración del juego cada agente puede modificar el comportamiento de sus correspondientes unidades.

4.1.5. Proyectil

El proyectil se crea desde la posición de la unidad, y la instancia del mismo se guarda en el objeto juego hasta su destrucción. Cada proyectil se mueve en la dirección indicada hasta que complete la distancia definida por el alcance de la unidad, o bien colisione con una unidad enemigo o con el Mapa. Los atributos del proyectil son los siguientes:

- **Posición:** indica la posición del proyectil, cuando es creado su posición será la de la unidad que realiza el disparo.
- **Dirección:** vector de dirección que indica hacia donde se va a dirigir el proyectil desde su lanzamiento hasta su impacto o límite de alcance.
- **Velocidad:** valor de la distancia recorrida por iteración de juego en la dirección indicada.
- **Daño:** señala el daño que realiza dicho proyectil al impactar sobre una unidad enemiga, dicho valor se asume de las características de la unidad que dispara el proyectil, esta característica permite que el sistema sea escalable a distintas unidades en un futuro.
- **Radio:** tamaño del proyectil que afecta a las colisiones con unidades y celdas del Mapa.

4.1.6. Agente

El objeto Agente es posiblemente el más complejo de todos. Se encarga de evaluar el código subido por los usuarios que servirá de agente, comprobar que la estructura del mismo es correcta y no ocasiona error a la hora de ejecutar, además de evaluar posibles comportamientos maliciosos. El Agente se encarga de ejecutar el código de los usuarios en un entorno controlado (Sandbox, que se explicará en otro punto más adelante), este proceso será llamado en dos etapas, la primera será en la inicialización del juego donde se le pasará el estado del juego. Posiciones de todas las unidades y el objeto Mapa. Y la segunda será llamado en cada iteración donde se le pasará un estado actualizado del juego, además de varias funciones y objetos útiles explicados en la API para el usuario. El agente deberá devolver, antes del tiempo de ejecución limitado por iteración, una lista de acciones para las unidades de su equipo. Posteriormente estas acciones serán procesadas y enviadas a través del objeto Juego a sus respectivas unidades. Si no se responde nada en un periodo de tiempo limitado se perderán las acciones para ese instante de juego.

4.1.7. Runner

Este objeto se encarga de gestionar todas las partidas encoladas en la aplicación. Instancia un objeto juego y se pone en marcha hasta finalizar la partida. Una vez se hayan guardado los resultados en base de datos se prosigue ejecutando partidas en orden hasta finalizar la cola de peticiones de partidas.

4.1.8. Motor de Físicas

El objeto motor de físicas se encarga de calcular en cada iteración de juego todas las colisiones entre los elementos del juego. Por un lado se calculan colisiones de las unidades con el mapa, creando una matriz de colisiones en círculo alrededor de la unidad en función a su radio. El valor de número de puntos de esta matriz de colisión es modificable por lo que se puede incrementar para una mayor precisión en las

colisiones, hecho esto se comprueba si cada uno de los puntos colisiona con un elemento del mapa. En caso afirmativo la unidad no realizará el último movimiento que provocó una colisión. Este cálculo es muy poco costoso dado que la matriz de colisiones está formada por un conjunto de vectores relativos a la posición de la unidad que se guardan en memoria. Dada la estructura del mapa, comprobar la colisión de estos puntos con el mapeado tiene un coste muy bajo. Véase en la figura 4.1.8, que está formado por 8 puntos de colisión.

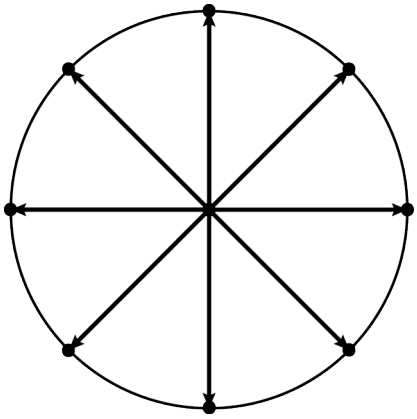

Figura 4.4: Matriz de colisiones de la unidad

Una vez generada la matriz de colisiones relativa a la unidad, formada por vectores que forman una circunferencia de N puntos alrededor del eje de coordenadas (0,0), para calcular las colisiones con el mapa en cada iteración se debe sumar a todos los elementos de esta matriz la posición de la unidad, obteniendo una matriz de puntos que rodean la unidad a la distancia de su radio. Una vez hecho esto se comprueba si cada uno de los puntos colisiona con alguna celda del mapa, tal y como se muestra en la figura 4.1.8, uno de los 8 puntos de la matriz de colisiones se encuentra dentro de una casilla del mapa marcada como zona sólida y por lo tanto en este caso se tratará de una colisión con el mapa.

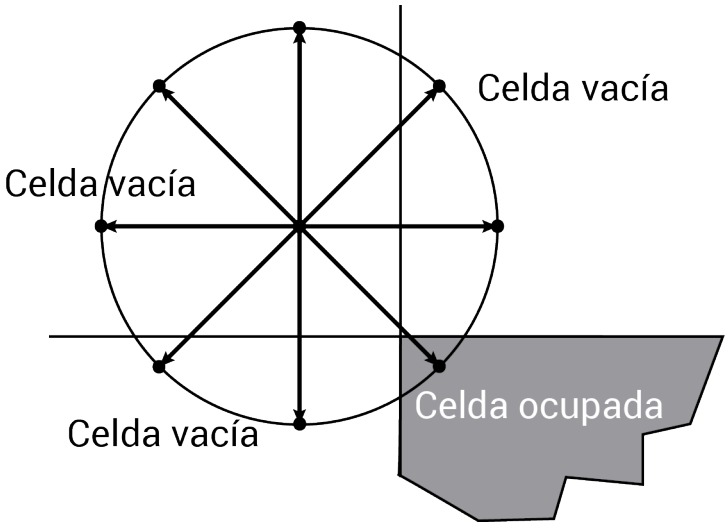

Figura 4.5: Colisión de una unidad con el mapa

Para las colisiones de los proyectiles con el Mapa se realiza el mismo proceso. Adicionalmente, los proyectiles también pueden colisionar con las unidades. Dicho cálculo se realiza comprobando que la

distancia de un proyectil a las unidades enemigas es menor que la suma de los radios del proyectil y la unidad, resultando así en una colisión.

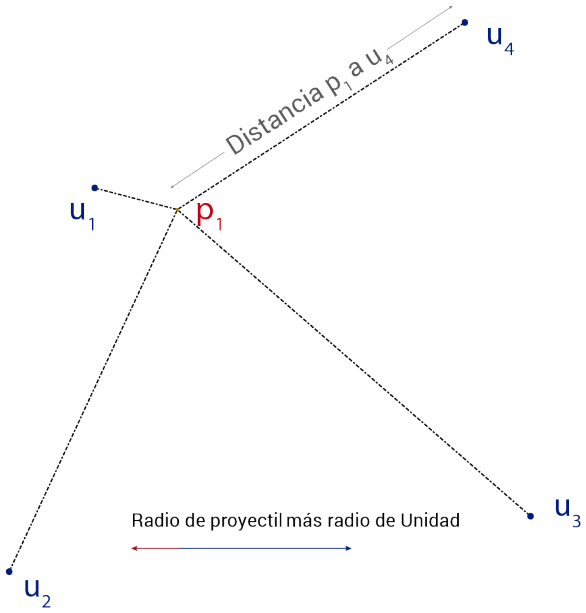

Figura 4.6: Colisión de una unidad con el mapa

4.2. Requisitos del sistema

En esta sección se tratarán los requisitos que debe tener el sistema final implementado. Los requisitos se han clasificado en función al apartado que hacen referencia. El formato a seguir por los requisitos es el siguiente:

- **Identificador:** identificador del requisito. Debe tener un formato R-[Tipo]-[Id], donde:
  - **Tipo:** es el código que identifica a que objeto afecta dicho requisito.
  - **Id:** identificador del requisito incremental, cada tipo tiene su numeración propia.
- **Prioridad:** indica la prioridad del requisito en función a las necesidades del mismo y tiempo de implementación en el sistema, están clasificadas en alta, media y baja.
- **Tipo de Requisito:** indica si el requisito es funcional (define algo que el sistema. debe permitir) o no funcional (define algo que el sistema no debe permitir).
- **Descripción:** texto descriptivo sobre la utilidad del requisito.
- **Subsistema:** máquina virtual o servidor.
- **Dependencias:** requisitos del que depende este requisito.
- **Dependiente:** requisitos que son dependientes de este requisito.

Identificador	**R-[Tipo]-[Id]**
Prioridad	[Alta | Media | Baja]
Tipo de Requisito:	[Func. | No Func.]
Descripción	[Descripción]
Subsistema	[Engine Core | Sandbox]

Tabla 4.1: Plantilla requisitos

4.3. Requisitos generales

Identificador	**R-G-0001**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	El sistema debe estar desarrollado en javascript, para poder ser ejecutado en un servidor Node.js
Subsistema	Engine Core

Tabla 4.2: R-G-0001

Identificador	**R-G-0002**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	El código subido por los usuarios debe compilar sin errores, además de tener implementadas las funciones básicas definidas en la API
Subsistema	Engine core

Tabla 4.3: R-G-0002

Identificador	**R-G-0003**
Prioridad	Media
Tipo de Requisito:	Func.
Descripción	El sistema debe ser capaz de leer mapas de base de datos
Subsistema	Engine Core

Tabla 4.4: R-G-0003

Identificador	**R-G-0004**
Prioridad	Media
Tipo de Requisito:	Func.
Descripción	El sistema debe ser capaz de importar mapas en base de datos
Subsistema	Engine Core

Tabla 4.5: R-G-0004

Identificador	**R-G-0005**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	El sistema debe ser capaz de ejecutar una partida
Subsistema	Engine Core

Tabla 4.6: R-G-0005

Identificador	**R-G-0006**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	El sistema debe ser capaz de guardar el resultado de la partida en base de datos
Subsistema	Engine Core

Tabla 4.7: R-G-0006

Identificador	**R-G-0007**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	El sistema debe ser capaz de ejecutar el código de los agentes en una Máquina Virtualpasando como parámetro el estado del juego y recibiendo del agente las acciones para esa iteracción
Subsistema	Engine Core

Tabla 4.8: R-G-0007

Identificador	**R-G-0008**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Si se recibe una petición de partida y el sistema está ocupado, debe encolar la petición de partida para ejecutarla más tarde.
Subsistema	Engine Core

Tabla 4.9: R-G-0008

Identificador	**R-G-0009**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	El sistema debe proveer las funciones definidas en la API y que sean accesibles por el agente en la Máquina Virtual, pathfinding A*, acceso a Vector2D, etc.
Subsistema	Engine Core

Tabla 4.10: R-G-0009

Identificador	**R-G-0010**
Prioridad	Media
Tipo de Requisito:	Func.
Descripción	Las unidades deben colisionar con el Mapa, evitando así que puedan atravesar muros o salirse del mismo.
Subsistema	Engine Core

Tabla 4.11: R-G-0010

Identificador	**R-G-0011**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Las unidades pueden moverse si se recibe dicha acción del agente.
Subsistema	Engine Core

Tabla 4.12: R-G-0011

Identificador	**R-G-0012**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Las unidades pueden disparar si se recibe dicha acción del agente.
Subsistema	Engine Core

Tabla 4.13: R-G-0012

Identificador	**R-G-0013**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Los proyectiles se mueven en una dirección fija determinada por la unidad que realiza el disparo.
Subsistema	Engine Core

Tabla 4.14: R-G-0013

Identificador	**R-G-0014**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Los proyectiles impactan con el Mapa y son destruidos si eso ocurre.
Subsistema	Engine Core

Tabla 4.15: R-G-0014

Identificador	**R-G-0015**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Los proyectiles impactan con unidades enemigas y son destruidos, además la unidad enemiga impactada recibe daño.
Subsistema	Engine core

Tabla 4.16: R-G-0015

Identificador	**R-G-0016**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	La unidad que recibe un impacto del proyectil recibe daño.
Subsistema	Engine Core

Tabla 4.17: R-G-0016

Identificador	**R-G-0017**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Si una unidad que recibe daño su vida baja a un valor inferior o igual a 0, la unidad muere.
Subsistema	Engine Core

Tabla 4.18: R-G-0017

Identificador	**R-G-0018**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Las unidades que están muertas no tienen efecto en el entorno de juego.
Subsistema	Engine Core

Tabla 4.19: R-G-0018

Identificador	**R-G-0019**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Cuando sólo queda un equipo con unidades vivas la partida finaliza.
Subsistema	Engine Core

Tabla 4.20: R-G-0019

Identificador	**R-G-0020**
Prioridad	Alta
Tipo de Requisito:	Func.
Descripción	Cuando el tiempo límite de la partida se acaba la partida finaliza.
Subsistema	Engine Core

Tabla 4.21: R-G-0020

4.4. Casos de Uso

En esta sección se van a definir los casos de uso, que servirán para dejar más claro las funcionalidades que el sistema tendrá que implementar.

4.4.1. Identificación de los actores

Los actores de los casos de uso serán los elementos que interactúan con la aplicación:

- **Runner:** se encarga de gestionar las partidas, ejecuciones, etc.
- **Juego:** esta a cargo de gestionar todo el flujo de la partida.
- **Agente:** obtiene el estado del juego y devuelve una acción.

El siguiente diagrama de casos de uso mostrado en la figura 4.4.1 , representa todos los casos de uso de la aplicación.

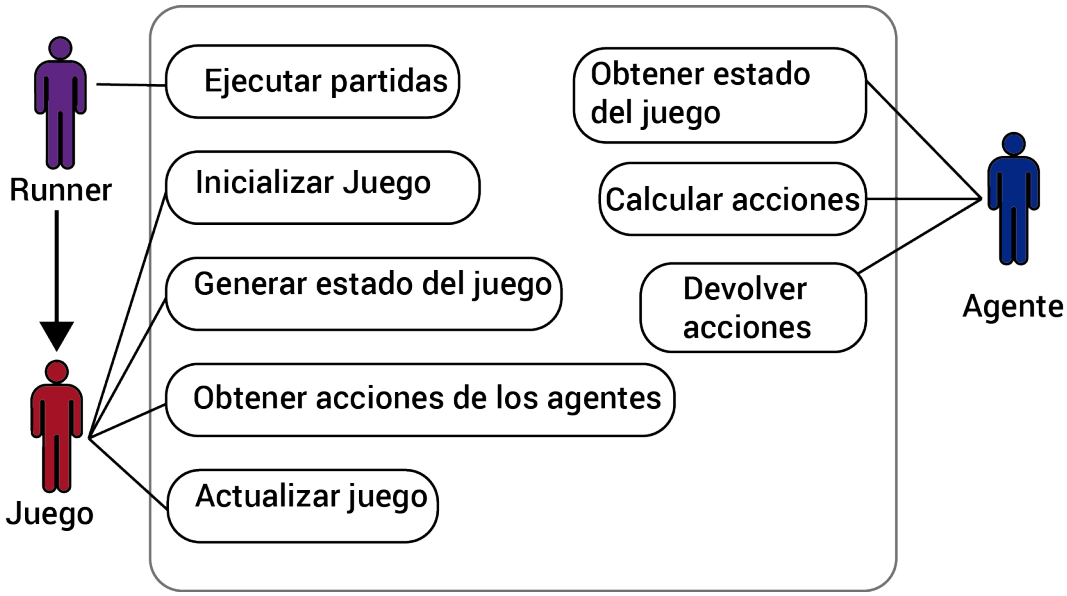

Figura 4.7: Diagrama casos de uso del motor de juego

La plantilla utilizada para los casos de uso será la mostrada en la siguiente tabla 4.22.

Nombre	**[Identificador caso de uso] - [Nombre descriptivo]**
Descripción	[Descripción resumida del caso de uso]
Actores	[Actor implicado en el caso de uso]
Precondiciones	[Dependencias del caso de uso]
Postcondiciones	[Dependen de este caso de uso]
Secuencia	[Descripción detallada de la secuencia de acciones que implica este caso de uso]
Secuencia Alternativa	[Explicación detallada de casos alternativos que dependen su ejecución de cierta condición]

Tabla 4.22: Plantilla casos de uso

A continuación se muestran los casos de uso de la aplicación.

Nombre	**Caso 1 - Ejecutar partidas**
Descripción	El agente Runner se encarga de ejecutar partidas cuando hay alguna en la cola para ejecutar.
Actores	Runner
Precondiciones	Ninguna
Postcondiciones	Ninguna
Secuencia	**1.** El actor Runner comprueba si hay partidas para ejecutar en la cola. **2.** En caso afirmativo pasa los parámetros a Juego y realiza la ejecución de la partida. **3.** Espera a que la partida finalice. **4.** Una vez finalizada la partida se vuelve a comprobar si hay partidas listas para ejecutar en la cola.
Secuencia Alternativa	Ninguna

Tabla 4.23: Caso 1 - Ejecutar partidas

Nombre	**Caso 2 - Inicializar Juego**
Descripción	El actor Juego inicializa el ciclo de una iteración en el motor de juego.
Actores	Juego
Precondiciones	Ejecutar partida
Postcondiciones	Ninguno
Secuencia	Se crea una instancia del juego, que consiste en las siguientes fases: **1.** Inicializar mapa del juego, se carga y procesa el mapa para la partida **2.** Crear equipos y asociarlos a sus respectivos agentes que los controlan, que a su vez se cargan desde base de datos. **3.** Se crean y posicionan las unidades de cada equipo en el mapa. **4.** Se inicializa el motor de físicas. **5.** Se calcula el estado del juego.
Secuencia Alternativa	Ninguna

Tabla 4.24: Caso 2 - Inicializar Juego

Nombre	**Caso 3 - Generar estado del juego**
Descripción	Obtiene el estado actual del juego en formato JSON
Actores	Juego
Precondiciones	Inicializar Juego
Postcondiciones	Obtener estado del juego
Secuencia	Obtiene información del estado actual de la partida y almacena la información en formato JSON, obtiene los datos en el siguiente orden: **1.** Obtener los datos del mapa, dimensiones del mismo, array de colisiones, etc. **2.** Obtiene los datos de los equipos y de sus respectivas unidades **3.** Se crean y posicionan las unidades de cada equipo en el mapa. **4.** Obtiene los datos de los proyectiles que hay en el juego, posición, dirección y velocidad.
Secuencia Alternativa	Ninguna

Tabla 4.25: Caso 3 - Generar estado del juego

Nombre	**Caso 4 - Obtener acciones de los agentes**
Descripción	Obtiene las acciones lanzadas por los agentes de la partida y las procesa.
Actores	Juego
Precondiciones	Devolver acciones
Postcondiciones	Actualizar juego
Secuencia	Recibe el conjunto de acciones enviadas por los agentes y las procesa. **1.** Se comprueba que el formato en el que se han recibido las acciones es el correcto. **2.** Se procesa una acción por cada unidad de los dos equipos **3.** Las acciones recibidas pueden ser de movimiento, de ataque o una combinación de ambas **4.** Se asignan las acciones recibidas a las unidades respectivas en la partida.
Secuencia Alternativa	En caso de que un agente no haya utilizado el formato correcto para las acciones de sus unidades: **1.** Se invalidan las acciones recibidas en ese turno.

Tabla 4.26: Caso 4 - Obtener acciones de los agentes

Nombre	**Caso 5 - Actualizar Juego**
Descripción	Ejecuta las acciones de las unidades y actualiza las físicas del juego.
Actores	Juego
Precondiciones	Obtener acciones de los agentes
Postcondiciones	Obtener el estado del juego
Secuencia	Realiza una iteracción del juego actualizando los siguientes elementos: **1.** Realiza el movimiento de las unidades, y comprueba colisiones de las mismas **2.** Realiza el ataque de las unidades, y actualiza las posiciones de los proyectiles en juego. **3.** Calcula las colisiones de los proyectiles contra las unidades y el mapa. **4.** Se destruyen las balas que han colisionado.
Secuencia Alternativa	Si en el paso 3, el proyectil colisiona con esa unidad: **1.** Decrementa la vida de la unidad en la cantidad de daño del proyectil. **2.** Si la vida de la unidad llega a cero cambia el estado de la unidad a muerta.

Tabla 4.27: Caso 5 - Actualizar Juego

Nombre	**Caso 6 - Obtener estado del juego**
Descripción	El agente obtiene el estado del juego
Actores	Agente
Precondiciones	Generar estado del juego
Postcondiciones	Calcular acciones
Secuencia	Obtiene el estado del juego pasado por el actor Juego como parámetro.
Secuencia Alternativa	Ninguna

Tabla 4.28: Caso 6 - Obtener estado del juego

Nombre	**Caso 7 - Calcular acciones**
Descripción	El agente calcula las acciones para sus unidades en funcion del estado del juego
Actores	Agente
Precondiciones	Obtener estado del juego
Postcondiciones	Devolver acciones
Secuencia	El agente a partir del estado del juego calculas las acciones a realizar: **1.** Calcula la heurística del estado actual. **2.** Realiza pathfinding a las posiciones de los enemigos. **3.** Mediante un algoritmo de inteligencia artificial se define la mejor acción para las unidades.
Secuencia Alternativa	Ninguna

Tabla 4.29: Caso 7 - Calcular acciones

Nombre	Caso 8 - Devolver acciones
Descripción	El Agente devuelve las acciones al Juego
Actores	Agente
Precondiciones	Calcular acciones
Postcondiciones	Obtener acciones de los agentes
Secuencia	Genera el conjunto de acciones para sus unidades en formato legible para el actor Juego
Secuencia Alternativa	Ninguna

Tabla 4.30: Caso 8 - Devolver acciones

4.5. Diseño

En esta sección se va a definir el diseño del sistema, que define la estructura del proyecto. Primeramente se crea la arquitectura del sistema, para posteriormente analizar cada uno de los subsistemas que la componen en profundidad.

4.5.1. Arquitectura de la aplicación

En esta sección se va a definir los patrones arquitectónicos y el diseño de la arquitectura.

Patrones arquitectónicos

A continuación se van a examinar los patrones estructurales de arquitectura comunes, a partir de los cuales se escogerá el más apropiado para el proyecto.

- **Programación por capas**: una arquitectura de cliente - servidor con objetivo principal de separar la lógica de negocios de la lógica de diseño. Se separa la capa de datos de la capa de presentación al usuario.
- **Modelo vista controlador(MVC):** se separan los datos, la lógica de negocio y la interfaz de usuario en módulos independientes.
- **Arquitectura orientada a servicios:** permite crear funcionalidad para diversos clientes que se comunican mediante un protocolo con la aplicación.

Los patrones arquitectónicos descritos pueden ser una buena opción para otro tipo de software, pero en este caso aplicar un modelo de descomposición funcional es el que más se adapta a las necesidades del proyecto.

4.5.2. Diseño detallado

En esta sección se detalla la estructura que tendrá el motor de juego, usando diagramas de clases UML.

Subsistema Motor de Juego

La clase central del sistema que se encarga de ejecutar las partidas es Game, a partir de ahí en la siguiente figura 4.5.2 se puede ver el resto de dependencias que forman el conjunto del motor de juego.

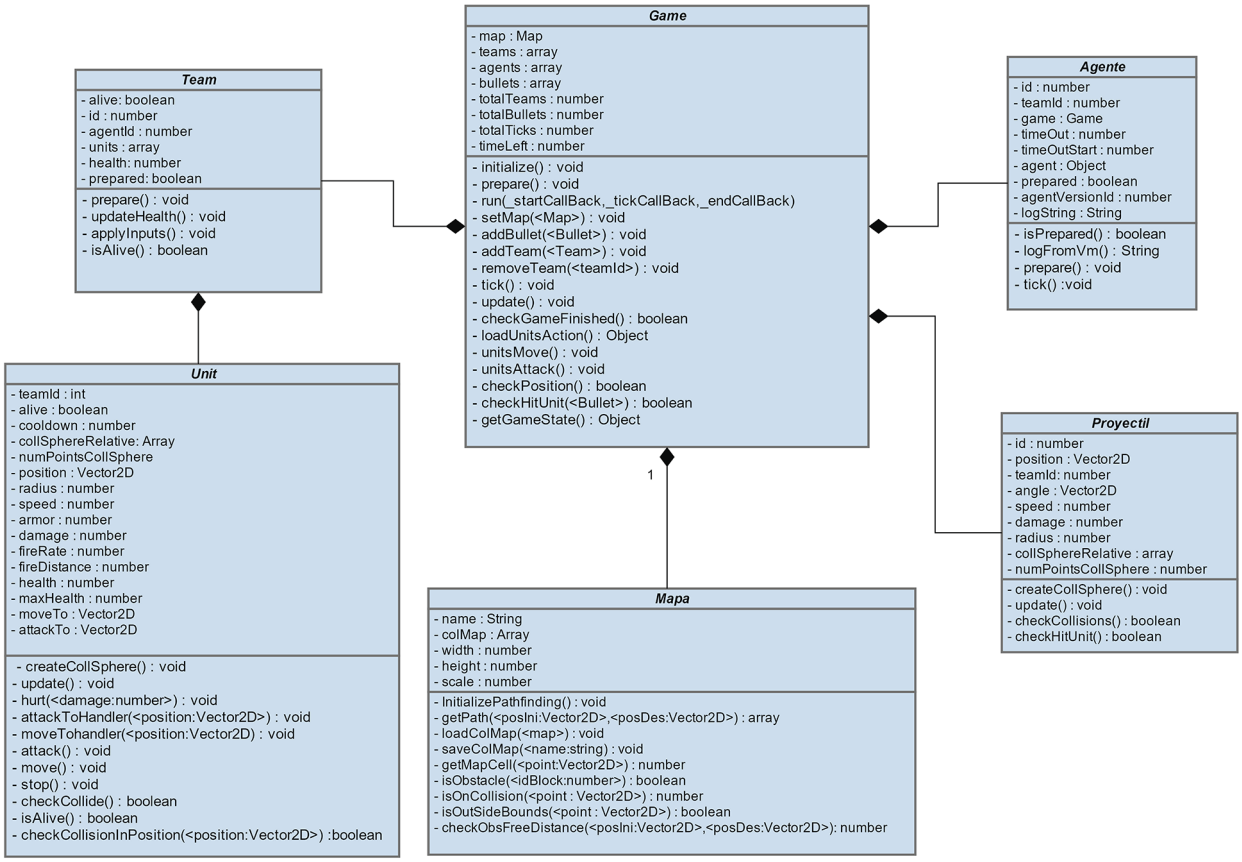

Figura 4.8: Diagrama UML del motor de juego

4.6. Máquina Virtual

La máquina virtual se utiliza para ejecutar el código subido por los usuarios en un entorno controlado y sin acceso al resto del sistema, esto garantiza la seguridad del sistema a la vez que evita que un código mal generado o que requiera más tiempo de cómputo del que se tiene asignado colapse todo el sistema.

Capítulo 5

Pruebas del Motor de juego

En este capítulo se detallará los resultados obtenidos con el motor de juego, primeramente se realizará un análisis sobre los problemas surgidos durante el desarrollo y cómo se han solucionado, después se realizará un análisis de rendimiento y pruebas de ejecución.

5.1. Corrección de errores

Durante todo el proceso de creación del motor de juego han surgido varios problemas de diversa índole, de todos ellos se van a destacar los que han tenido una mayor repercusión:

5.1.1. Comportamientos anómalos

Sobre todo al principio del desarrollo, cuando se comenzaron a realizar pruebas de funcionamiento se podía observar que las unidades respondían a las acciones correctamente al inicio de la partida, pero a medida que avanzaba el juego ocurrían cosas extrañas, unidades que desaparecían o cambiaban de posición y algunas unidades que habían muerto se quedaban estáticas en el mapa sin desaparecer.

Analizando en profundidad estos problemas se observaron varios fallos, el primero debido a las peculiaridades de javascript al llamar a funciones recursivas o mediante callbacks se perdía el contexto, que provocaba errores fáciles de detectar en el log del servidor.

El segundo fallo detectado fue que a la hora de realizar el ciclo de juego con las unidades y el motor de físicas, a la hora de enviar la posición por parámetro al motor de físicas y este después realizaba operaciones con el mismo, al estar pasando de manera errónea el parámetro por referencia se modificaba el valor de la posición de la unidad. El problema se solucionó pasando por parámetro siempre copias de los objetos, para así evitar que ocurriese en un futuro dada la dificultad de encontrar el fallo en las trazas.

Por último destacar como comportamiento anómalo unidades y balas que se quedaban estáticas en el juego sin desaparecer. En resumen el problema era almacenar todas las unidades y proyectiles en arrays sin tener un identificador de los mismos, por ese motivo a la hora de añadir y eliminar posiciones de los

arrays, las posiciones de los mismos se veían alteradas respecto a su identificador. La solución consistió en almacenarlos en objetos con el formato clave valor.

5.1.2. Problemas de ejecución en Máquina Virtual

Una vez el motor de juego funcionaba de forma estable, se realizó la implementación de máquinas virtuales para ejecutar el código subido. De esta nueva funcionalidad surgieron nuevos problemas:

Cargar scripts

Para cada ciclo del juego se llama a la máquina virtual para cargar el código de los agentes y obtener la respuesta en función al estado del juego que se le envía.

La implementación era poco óptima y se cargaba en memoria el código de los agentes una vez por iteración de juego. Produciendo que las partidas tardasen diez veces más en ejecutar y saturando la memoria del servidor.

Como solución se cargan los scripts en memoria al principio de la partida y posteriormente se realizan las llamadas directamente a memoria.

Trazabilidad errores de los agentes

El último error destacable con respecto a las máquinas virtuales es que un fallo de ejecución en los agentes no se veía reflejado en las trazas del servidor. Además los usuarios finales no tendrían acceso a las trazas del servidor, por lo que la solución más optima fue crear una variable para guardar los logs ocasionados en el entorno de la máquina virtual.

5.1.3. Problemas de rendimiento

Por último destacar un fallo de diseño, al principio se enviaba en cada iteración a los agentes una copia del mapa. Esto estaba diseñado así por si el mapa cambiaba por el tiempo, pero dado que el mapa es estático de momento no tiene mucho sentido. La solución consistió en dejar accesibles desde los agentes la variable mapa en una posición de memoria y pasada por parámetro a las máquinas virtuales.

5.2. Pruebas de funcionamiento

Para comprobar el correcto funcionamiento del motor de juego se han diseñado unos casos de prueba, siguiendo el formato de la plantilla 5.1, se define un evento y el resultado esperado.

Nombre	**[Identificador caso de prueba] - [Nombre descriptivo]**
Descripción	[Descripción del caso de prueba]
Evento	[Detalles del evento]
Resultado	[Resultado esperado]

Tabla 5.1: Plantilla casos de prueba

Nombre	**Caso 1 - Movimiento de la unidad**
Descripción	La unidad que recibe la orden debe moverse en la dirección correcta.
Evento	Se recibe una orden de movimiento para la unidad.
Resultado	La unidad se mueve en la dirección introducida la distancia por iteración que muestra en su variable velocidad

Tabla 5.2: Caso 1 - Movimiento de la unidad

Nombre	**Caso 2 - Colisión con entorno**
Descripción	La unidad que se desplaza hacia un muro a una distancia menor que su velocidad debe colisionar.
Evento	Mover unidad hacía un muro estando pegado a el
Resultado	La unidad permanece estática, si atraviesa el muro se marca la prueba como un fallo

Tabla 5.3: Caso 2 - Colisión con entorno

Nombre	**Caso 3 - Lanzamiento de proyectil**
Descripción	La unidad que recibe una orden de atacar debe crear un proyectil en su ubicación si el cooldown de ataque es cero
Evento	Recibir orden de ataque y cooldown de la unidad a cero.
Resultado	Se genera una bala en la ubicación de la unidad

Tabla 5.4: Caso 3 - Lanzamiento de proyectil

Nombre	**Caso 4 - Colisión de proyectil con mapeado**
Descripción	La bala debe impactar al colisionar con un muro. Eso implica que desaparecerá del juego
Evento	La trayectoria del proyectil entre dos instantes atraviesa un muro.
Resultado	El proyectil se elimina del juego.

Tabla 5.5: Caso 4 - Colisión de proyectil con mapeado

Nombre	**Caso 5 - Proyectil impacta a una unidad**
Descripción	El proyectil impacta a una unidad enemiga.
Evento	El proyectil entre dos instantes se encuentra a una distancia menor que la suma de su radio al de la unidad.
Resultado	La unidad es dañada y el proyectil destruido.

Tabla 5.6: Caso 5 - Proyectil impacta a una unidad

Nombre	**Caso 6 - Matar a una unidad**
Descripción	Una unidad dañada recibe daño que baja su salud a cero.
Evento	La unidad recibe más daño que vida actual.
Resultado	La unidad queda marcada como muerta y su indicador de vida a cero.

Tabla 5.7: Caso 6 - Matar a una unidad

Nombre	**Caso 7 - Todas las unidades de un equipo están muertas**
Descripción	Todas las unidades de un equipo mueren
Evento	Muere la última unidad de un equipo.
Resultado	Finalizar la partida marcando como a ganador al equipo rival.

Tabla 5.8: Caso 7 - Todas las unidades de un equipo están muertas

Nombre	**Caso 8 - El tiempo finaliza**
Descripción	El tiempo de la partida llega a su fin.
Evento	El contador de tiempo restante llega a cero.
Resultado	La partida finaliza, dejando como ganador al equipo con mas vida total, en caso de igualar vida quedaría en empate.

Tabla 5.9: Caso 8 - El tiempo finaliza

5.3. Tiempos de ejecución de partidas

Dadas las características del sistema, las partidas son ejecutadas en servidor y después se pueden ver a través del visualizador de partidas implementado por el cliente. La partida se reproduce a 60 fotogramas por segundo, eso implica que una partida de dos minutos tiene un total de siete mil doscientos frames.

Por suerte la ejecución de las partidas es muy inferior a el tiempo de visualización, eso permite que se puedan ejecutar varias partidas a la vez, sin que se genere mucha demora para mostrar los resultados. A continuación se muestra una tabla con los tiempos medios en las partidas ejecutadas de la batería de pruebas.

Agente	**Cantidad**
Partidas	210
Frames	441.317
Tiempo de ejecución	635
Tiempo medio por partida	3,023 segundos

Tabla 5.10: Tiempos ejecución de partidas

Capítulo 6

Desarrollo del Agente

Una vez funcionando el motor del juego que se encargará de ejecutar las partidas y el código proporcionado por los agentes, lo único que queda es crear un agente inteligente para probar la plataforma. En esta sección se va a realizar un análisis del problema para poder escoger un algoritmo que se adapte a sus necesidades y sea capaz de responder de forma óptima en este entorno.

6.1. Algoritmos de Pathfinding

El cálculo de la mejor ruta para las unidades es crucial para obtener el mejor resultado con el menor coste. Hay que tener en cuenta que la mejor ruta no es necesariamente la más corta, puede haber obstáculos en medio o áreas con un mayor coste para la unidad.

- **Dijkstra:** Algoritmo basado en grafos, cada nodo representa los diferentes puntos o celdas que las unidades pueden alcanzar, y las aristas definen el coste de moverse de un punto a otro. El algoritmo recorre todas las rutas posibles de un punto a otro calculando el coste y devuelve la ruta con menor coste obtenida.

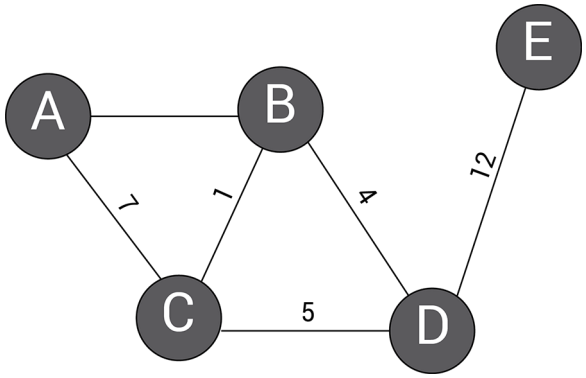

Figura 6.1: Diagrama algoritmo Dijkstra

- **Algoritmo de búsqueda A*:** Realiza una búsqueda al menor coste entre dos puntos. A diferencia del algoritmo de Dijkstra, hace uso de una función heurística que no sólo tiene en cuenta el coste del punto de origen, sino que además hace una estimación del coste al siguiente nodo. El algoritmo de búsqueda A* es el más usado actualmente en resolución de problemas de pathfinding. Obtiene mejores resultados que Dijkstra y además reduce el espacio de búsqueda al calcular la distancia al siguiente punto. En la figura 6.1 se observa el cálculo de coste desde el punto A, hasta el punto B.

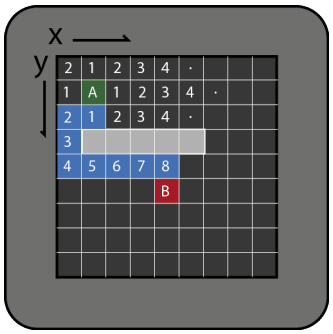

Figura 6.2: Diagrama algoritmo de búsqueda A*

- **A* en tiempo real:** Modificación del algoritmo de búsqueda A* para aplicaciones en dominios de tiempo real. Guarda el valor de cada estado visitado en una tabla y la mantiene actualizada mediante el uso de técnicas de programación dinámica.

6.2. Algoritmos Inteligencia artificial

En esta sección se analizaran los diferentes tipos de Algoritmos de IA para toma de decisiones, posteriormente se escogerá el que mejor se adapte al ámbito del problema y se utilizará para implementar un agente en la plataforma.

- **Lógica difusa:** puede ser definido como un tipo de lógica que hace uso de la incertidumbre para toma de decisiones, tratando de imitar el comportamiento humano de pensar. En la lógica clásica las sentencias son verdaderas o falsas, sin embargo en la lógica difusa hay varios grados de verdad. Las decisiones en lógica difusa son tomadas acorde a un arbol de decisiones que está asociado a diferentes estados difusos.

- **Algoritmo de la colonia hormigas:** tiene su origen en 1992, propuesto por Marco Dorigo en su tesis de doctorado. La idea original era resolver problemas de camino óptimos en grafos, pero actualmente es aplicado a todo tipo de problemas de diferente índole.

- **QLearning:** también conocido como aprendizaje por refuerzo, se analizan las acciones del agente y su entorno, y en función del resultado de esas acciones se otorga una recompensa que refuerza los resultados positivos y penaliza los negativos. Para ello hay que escoger las variables del entorno que sean relevantes para la toma de decisiones, y realizar una función heurística para el cálculo de la recompensa. A partir de estos dos factores, primeramente se realizan múltiples ejecuciones intentando conseguir la mayor diversidad en los datos del entorno, a partir de los cuales se obtendrán los estados mediante clusterización. Una vez se tienen los estados calculados hay que entrenar el

agente con partidas para que mediante la recompensa genere y actualice la matriz Q, que es la encargada de calcular para que estado es la mejor acción.

La implementación de un algoritmo de toma de decisiones basado en Qlearning puede ofrecer buenos resultados en un problema de complejidad NP-Hard.

6.3. Diseño del Agente

En esta sección se describe como realizar la implementación del agente, utilizará un algoritmo de decisiones QLearning, y para el cálculo de rutas evitando obstáculos hará uso del algoritmo de pathfinding A* en tiempo real.

A continuación se van a describir los pasos llevados a cabo para implementar Qlearning al agente. Dadas las peculiaridades de la plataforma, para una primera aproximación las decisiones se van a calcular de manera independiente por unidad, esto otorga varias ventajas, que son entre otras, tener muchos datos en el proceso de aprendizaje al poder recopilar datos de cada unidad en cada instante de tiempo, si la partida tiene 10 unidades se obtendrán 10 veces más datos por partida que tomando una estrategia de toma de estados por equipo. La principal desventaja es que las unidades no trabajarían en equipo ni se llegaría a desarrollar una estrategia común.

Primeramente se analizan las variables del entorno que se van a tener en cuenta a la hora de representar los estados de la unidad para la toma de decisiones.

6.3.1. Datos del estado

Para representar el estado actual de la unidad, se necesitan definir los parámetros o variables que son relevantes en torno a la unidad. De manera genérica se definirán todos los posibles para elegir los más relevantes que serán utilizados más adelante.

- **Unidades enemigas:** la información de las unidades enemigas en torno a la unidad es crucial para la toma de decisiones, las variables que vamos a utilizar son las siguientes:
  - **Posición:** se toma como dato la posición de la unidad enemiga en relación a la posición propia. De esta forma se tiene constancia de la distancia a la misma y su orientación hacia nuestra unidad.
  - **Vida:** la vida es un factor importante, indica lo difícil que resultaría matar esa unidad. Además si se compara con la vida de la unidad propia podría indicar si estamos en desventaja o no.
- **Proyectiles enemigos:** las unidades enemigas nos atacan, y generan proyectiles desde sus posiciones con el fin de producir daño sobre las unidades aliadas, es por eso que la información sobre los proyectiles cercanos es decisiva.
  - **Posición:** la posición relativa del proyectil en torno a la unidad.
  - **Dirección:** la dirección del proyectil indica si se dirige hacia nuestra unidad, viene dada por un vector normalizado.

- **Total unidades vivas enemigas:** el número total de unidades enemigas que siguen vivas nos sirve de indicador de la fuerza que tiene el enemigo, a mayor número de unidades más proyectiles simultáneos se podrán recibir.
- **Vida de la unidad propia:** indica lo débil que es la unidad, sin duda un factor decisivo para poder llevar una filosofía defensiva cuando le queda poca vida a la unidad.
- **Obstáculos cercanos:** por último no se van a dejar de lado el entorno, es muy importante tener en cuenta este factor.
- **Disparo en cooldown:** la unidad tan sólo obedecerá la orden de atacar si se ha terminado el tiempo de cooldown entre disparos. Es importante para que el agente pueda aprender a disparar correctamente disponer de la información de este valor.
- **Número de enemigos con visión directa:** otro factor muy importante es el número de enemigos que tienen visión directa con la unidad, conocer este valor puede ayudar a conseguir comportamientos como cobijarse detrás de muros para intentar disminuir dicha cantidad.

Comportamientos avanzados

- **Refugiarse tras un muro:** contabiliza el número de unidades enemigas que tienen visión directa con la unidad, utilizando ese valor para el cálculo de la heurística propiciará que la unidad se esconda.
- **Esquivar una bala:** realizar un conteo de las balas que iban a impactar a la unidad y que ha conseguido esquivar, podría ser de gran utilidad a la hora de promover que el agente aprenda a esquivar balas mejor

En la figura 6.3.1 se observa la unidad actual(verde), las unidades aliadas(azul) y las enemigas(rojo). Los proyectiles enemigos(amarillo) y su dirección(linea de puntos) hacia la unidad.

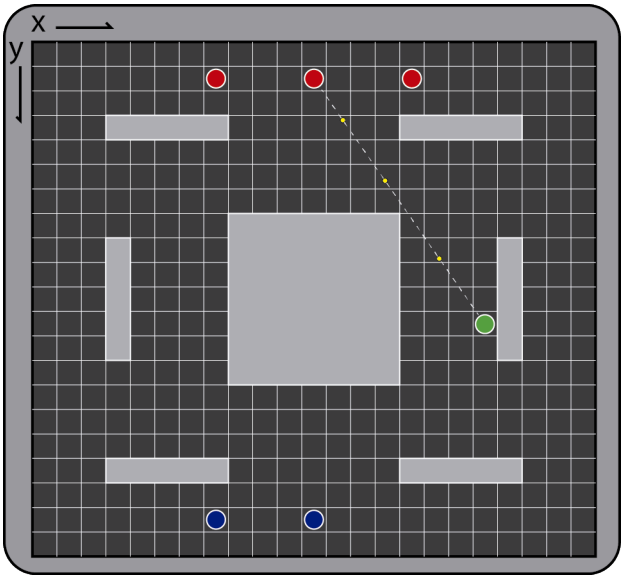

Figura 6.3: Estado del entorno de la unidad

6.3.2. Definición de las acciones

Las acciones disponibles de cada unidad consisten en movimiento y ataque, ambas se indican a través de una posición del mapa a la que se desea mover la unidad o realizar el ataque. Internamente esta tratado como un vector normalizado indicando una dirección, véase la figura 6.3.2.

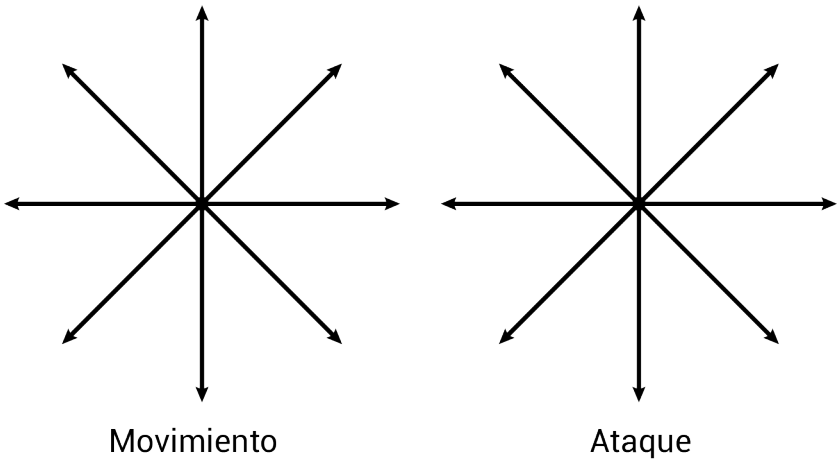

Figura 6.4: Vectores de movimiento y ataque

Tanto la acción de movimiento como disparo tiene muchos grados de libertad, y aún pudiendo reducirlo a tan sólo 8 para cada uno, quedando tal y como se ve en la figura anterior, eso ocasionaría varios problemas, el primero es que se perdería precisión en los disparos y el segundo es que para codificar la acción con los dos sería una combinación de nueve posibles direcciones de movimiento (la novena opción es quedarse quieto), y ocho posibles de disparo.

En total suman 72 posibles acciones, que es de poca utilidad para un aprendizaje por refuerzo. Requeriría de una cantidad inmensa de datos de entrenamiento y aún así no garantizaría su buen funcionamiento.

En lugar de eso, se puede codificar en un menor número de acciones que sean más genéricas. Cada una de esas acciones será programada para cumplir su función, como consecuencia el grado de libertad de la unidad se verá afectado reduciendo la posibilidad de que aprenda comportamientos nuevos, pero como ventaja será capaz de optimizar dichas acciones.

- **Búsqueda de enemigos(avanzar):** localiza al enemigo más próximo y realiza el cálculo de la ruta más corta mediante A* para realizar la aproximación.
- **Atacar:** atacar a la unidad más cercana con visión directa, en un principio se utilizará la posición enemiga para realizar el ataque, una vez probada su eficacia más adelante se podrán utilizar predicciones de movimiento enemigas para conseguir dar a unidades en movimiento.
- **Esquivar:** comprueba los proyectiles enemigos cercanos y los esquiva, para ello se realizará un calculo de posiciones de los proyectiles y trayectorias con sus vectores de dirección. La unidad debe esquivar de la manera más óptima.
- **Avanzar y atacar:** Avanza hacia el enemigo a la vez que realiza una acción de ataque.
- **Esquivar y atacar:** Realiza las acciones de esquivar y atacar al mismo tiempo.

6.3.3. Heurísticas

Para el aprendizaje por refuerzo utilizado en QLearning se necesita calcular una recompensa. En esta sección vamos a definir la heurística entre transiciones de estados.

Para realizar un correcto aprendizaje se necesita un estado, una acción, el estado siguiente y una recompensa. En los apartados anteriores se han definido los parámetros para calcular el estado y las diferentes acciones disponibles, a continuación definiremos el cálculo de la recompensa

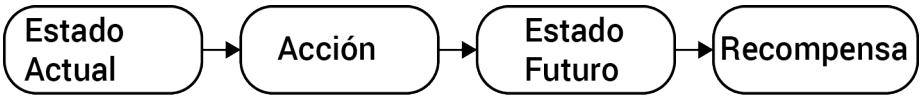

Figura 6.5: Tupla QLearning

A continuación se definen los parámetros tomados para el cálculo de la recompensa:

- **Unidad enemiga dañada:** Bonificación **+5**
- **Unidad enemiga muerta:** Bonificación **+15**
- **Unidad aliada dañada:** Penalización **-5**
- **Unidad aliada muerta:** Penalización **-15**
- **Acercarse al enemigo:** Bonificación **+1**

Estos parámetros modifican el aprendizaje y futuro comportamiento de nuestro agente. Realizando modificaciones sobre los mismos se conseguirán comportamientos diferentes.

Si se desea un comportamiento más defensivo bastaría con penalizar más por las unidades perdidas que por las matadas, se han decidido estos valores para un comportamiento más balanceado. Además al penalizar más por una unidad muerta que por quitar vida favorece que nuestras unidades colaboren y algunas unidades puedan interponerse en un proyectil que termine matando a un aliado.

Por otro lado se utiliza una bonificación de acercamiento al enemigo de **+1** para evitar que las unidades permanezcan quietas al inicio de la batalla. Se podría valorar un comportamiento más defensivo pero se desea que el agente tenga la iniciativa en el ataque, que en las primeras fases de entrenamiento y aprendizaje puedan aportar muchos más datos y estados útiles para el aprendizaje del agente.

A continuación se muestran varios ejemplos de aplicar esta heurística en la transición entre dos posibles estados.

En el primer ejemplo se daña a una unidad enemiga, la recompensa obtenida es de **+5**, por lo tanto recibirá refuerzo a la acción realizada y se verá reflejado en la matriz Q.

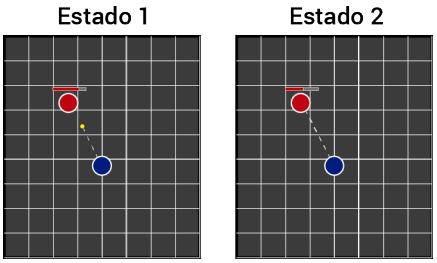

Figura 6.6: Ejemplo 1 - Dañar unidad enemiga

Motivo	**Bonifiación + / Penalización -**
Enemigo dañado	5
Enemigo muerto	-
Aliado dañado	-
Aliado muerto	-
Avanzar	-
Recompensa	**5**

Tabla 6.1: Ejemplo 1 - Dañar unidad enemiga

En el segundo ejemplo la unidad avanza y es eliminada por un proyectil enemigo, en este caso se obtiene **+1** por avanzar y **-15** por la muerte de la unidad. Durante el entrenamiento detectará la acción de avanzar contra un proyectil enemigo como mala y será penalizado en la matriz Q.

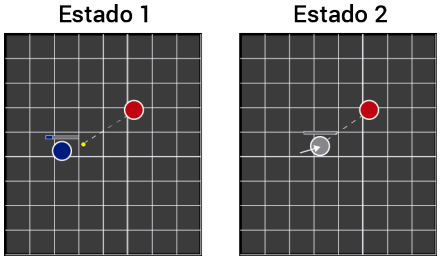

Figura 6.7: Ejemplo 2 - Avanzar contra proyectil

Motivo	**Bonifiación + / Penalización -**
Enemigo dañado	-
Enemigo muerto	-
Aliado dañado	-
Aliado muerto	-15
Avanzar	+1
Recompensa	**-14**

Tabla 6.2: Ejemplo 2 - Avanzar contra proyectil

En el siguiente ejemplo una unidad se interpone en la trayectoria de la bala que va a impactar sobre un aliado a punto de morir, en este caso se obtiene una penalización de **-4** pero aún así después del aprendizaje se verá reflejado en la matriz Q esta acción como mejor opción que dejar que impacte sobre el aliado.

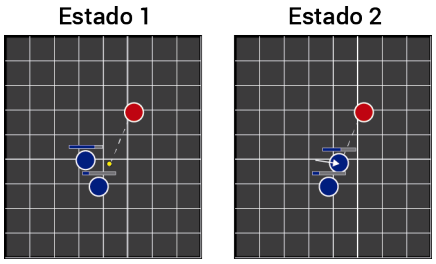

Figura 6.8: Ejemplo 3 - Salvar unidad aliada

Motivo	Bonificación + / Penalización -
Enemigo dañado	-
Enemigo muerto	-
Aliado dañado	-5
Aliado muerto	-
Avanzar	+1
Recompensa	**-4**

Tabla 6.3: Ejemplo 3 - Salvar unidad aliada

En el último ejemplo se puede observar una combinación de varias acciones, se avanza al enemigo mientras se dispara, matándolo y a la vez recibiendo un proyectil enemigo que daña la unidad. Sin embargo la penalización de recibir daño sin morir (**-5**) es menor que la de matar una unidad enemiga (**+15**), a lo que se suma **+1** por avanzar. Recibiendo una recompensa total de **+11**.

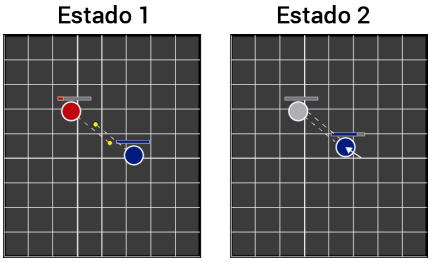

Figura 6.9: Ejemplo 4 - Dañar unidad enemiga

Motivo	Bonifiación + / Penalización -
Enemigo dañado	-
Enemigo muerto	+15
Aliado dañado	-5
Aliado muerto	-
Avanzar	+1
Recompensa	**11**

Tabla 6.4: Ejemplo 4 - Dañar unidad enemiga

En resumen, se puede observar que aunque las decisiones tomadas usando Qlearning se realicen a nivel de unidad por separado, mediante la heurística utilizada se favorece el trabajo en equipo. Se prevee que pueda dar buenos resultados a la hora de llevarlo a la práctica.

6.3.4. Clusterización

Para definir los estados se utilizarán los parámetros vistos en el apartado anterior 6.3.1, generando datos de partidas reales se creará un conjunto con dichos datos para posteriormente clusterizarlos generando los centroides que servirán para indicar los diferentes estados.

Para generar el conjunto de datos con partidas reales, se han creado diferentes agentes con el fin de crear la mayor variedad de comportamientos distintos, la idea es generar la mayor diversidad en los datos del estado para tener el mayor número posible de estados diferentes a los que se puede enfrentar nuestro agente. Los agentes creados son los siguientes:

Agente aleatorio

Consiste en un agente con acciones para todas sus unidades completamente aleatorias, esto aporta variedad en la generación de nuevos estados. Aunque se limitará el número de batallas de este agente debido a que al tener el mismo comportamiento siempre puede producir sobre aprendizaje.

Agente asesino

Este agente está diseñado para perseguir a sus victimas hasta acabar con ellas o morir en el intento, se le dota de A* para evitar obstáculos y persigue de forma directa a las unidades mas cercanas mientras las dispara.

Agente asesino impreciso

Una variante del anterior, sigue el mismo funcionamiento, pero se le añade un poco de aleatoriedad a sus movimientos de tal forma que uno de cada 3 movimientos es aleatorio, además se modifica el vector de disparo en un pequeño factor aleatorio. Esto permite nuevos estados, e incluso que pueda golpear a unidades enemigas en movimiento en casos que el anterior agente hubiese fallado.

Agente ninja

Dotado de un poco más de inteligencia, se acerca a sus victimas hasta cierta distancia, les dispara de forma precisa y además tiene en cuenta los proyectiles enemigos, esquivándolos en función al vector de dirección de los mismos.
Una vez creados los agentes, serán utilizados para luchar entre ellos y generar los datos necesarios para la clusterización y obtención de estados, la herramienta está preparada para ponerlos a combatir automáticamente, lo que simplifica mucho el proceso. En la siguiente tabla se muestra la cantidad de batallas utilizadas para generar los datos.

Agente	**Aleatorio**	**Asesino**	**Asesino imp.**	**Ninja**
Aleatorio	10	10	10	10
Asesino	-	20	20	20
Asesino imp.	-	-	50	50
Ninja	-	-	-	150

Tabla 6.5: Planificación batallas

Agente	**Total partidas**	**Porcentaje**
Aleatorio	25	11,9 %
Asesino	45	21,4 %
Asesino imp.	60	28,6 %
Ninja	80	38,1 %
Total	**210**	**100 %**

Tabla 6.6: Total partidas por agente

Una vez ejecutadas las partidas, obtenemos la información de las mismas de base de datos, la información relevante y utilizada para el aprendizaje será la obtenida en los frames de las partidas

generadas, a partir de los cuales se generarán las tuplas con los parámetros deseados para clusterizar. Para ello hay que procesar los datos, en este caso el uso de Qlearning está enfocado a la unidad, por lo que de cada Frame se pueden obtener 12 tuplas, una por cada unidad en la batalla, en la tabla 6.7 se puede observar los frames obtenidos en las partidas y las tuplas generadas a partir de los mismos.

Agente	**Cantidad**
Partidas	210
Frames	441.317
Tuplas	5.295.804

Tabla 6.7: Tuplas clusterización

A partir de las tuplas generadas, se utilizará la herramienta Weka para realizar la clusterización, los parámetros introducidos inicialmente utilizando SimpleKMeans con 100 nodos.

En la primera prueba tan sólo nos ha generado 64 nodos y muchos de ellos con un porcentaje de datos asignado igual o inferior al 1 %, por lo tanto son descartados y se procede a realizar otra vez el proceso cambiando la configuración.

Después de diferentes pruebas, se obtienen mejores resultados. La configuración utilizada es de 8 nodos y 1500 iteraciones, en la tabla 6.8 se puede observar el resultado.

Centroide	**Cantidad**	**Porcentaje**
Centroide0	4.190	21 %
Centroide1	277	1 %)
Centroide2	7.474	37 %
Centroide3	1.102	5 %
Centroide4	629	3 %
Centroide5	4.172	20 %
Centroide6	584	3 %
Centroide7	1.299	6 %
Centroide8	657	3 %

Tabla 6.8: Centroides

6.3.5. Entrenamiento

Una vez obtenidos los centroides ya se puede obtener el estado a partir de las tuplas, y por lo tanto se tiene todo lo necesario para realizar el entrenamiento, recordemos la figura 6.3.3.

A partir de las tuplas usadas para generar los centroides, se reutilizarán para realizar el aprendizaje, para ello se necesita un estado actual(calculado por el centroide más cercano), un estado futuro, en este caso tomaremos 10 iteracciones de lapso de tiempo entre estados, la acción realizada y la recompensa, que será calculada en función de las diferencias entre los dos estados.

Una vez está todo listo para realizar el aprendizaje, se configura el algoritmo Qlearning con un valor gamma de 0.8 y 1000 iteraciones, se procesan todas las tuplas siguiendo el formato anterior y se obtiene la matriz Q, que será utilizada por el agente para la toma de decisiones. Una vez el agente esté en funcionamiento se utilizarán sus propias partidas para actualizar la matriz Q de tal forma que constantemente estará aprendiendo.

Capítulo 7

Pruebas y Resultados del Agente

En este capítulo se van a detallar los resultados y comportamientos del agente detallado en el capítulo anterior.

7.1. Resultados

Después de realizar el entrenamiento se pone a prueba al agente haciéndolo competir con los demás agentes que fueron utilizados para su entrenamiento. A continuación se detallan los resultados obtenidos en cada uno de ellos.

7.1.1. Agente aleatorio

El agente creado se comporta de forma óptima contra el agente aleatorio, es capaz de eliminarlo sin recibir apenas daño. En la tabla 7.1 se muestran los resultados medios resultantes de ejecutar 10 partidas.

Agente	**Vida total por equipo (%)**
Qlearning	98,5 %
Aleatorio	0 %

Tabla 7.1: Qlearning vs aleatorio

Se puede observar que gana todas las batallas, y en alguna de ellas es herido, lo que provoca que la vida media en todas las partidas baje.

7.1.2. Agente asesino

La segunda prueba se realiza contra un oponente más agresivo. En este caso se realizan primeramente 10 partidos, obteniendo los siguientes datos:

Agente	Partidas ganadas
Qlearning	8
Asesino	2

Tabla 7.2: Qlearning vs asesino - 10 partidas

De diez partidas el agente enemigo ha vencido dos veces, dado que es un enemigo con comportamientos simples este valor es muy alto. Se procede a entrenar el agente Qlearning contra el asesino en durante 1000 partidas, después se realizan 100 partidas para evaluar resultados, los resultados finales son estos:

Agente	Partidas ganadas
Qlearning	95
Asesino	5

Tabla 7.3: Qlearning vs asesino - 100 partidas

Se observa que el entrenamiento ha dado sus frutos, se ha reducido las partidas perdidas de un 20 % a tan sólo un 5 %.

7.1.3. Agente asesino impreciso

Dada la similitud con el agente anterior, se enfrenta directamente al Qlearning con 100 partidas, los resultados obtenidos son estos:

Agente	Partidas ganadas
Qlearning	99
Asesino	1

Tabla 7.4: Qlearning vs asesino impreciso - 100 partidas

Los datos obtenidos cuadran con lo esperado, dado que este agente es muy similar al anterior, y además debido a su naturaleza pseudoaleatoria falla más ataques que el anterior.

7.1.4. Agente ninja

Por último se enfrenta nuestro agente a el más complejo de los utilizados en el entrenamiento. En una primera ronda los resultados son:

Agente	Partidas ganadas
Qlearning	35
Asesino	65

Tabla 7.5: Qlearning vs ninja - 100 partidas

El comportamiento del agente ninja, pese a ser realizado programáticamente sin usar inteligencia artificial demuestra tener un comportamiento muy agresivo y su algoritmo de esquivar proyectiles le hace ganar.

Al contrario, el agente Qlearning parece no dar buenos resultados, puede ser debido a un problema de identificación de estados. Esto puede ser debido a que los estados generados en la clusterización no contemplaban todos los posibles estados, y se esté asignando a un centroide erróneo, para solucionar este

fallo se debe volver a generar los centroides con los datos de las nuevas partidas realizadas, que puedan aportar nuevos centroides.

Tras un entrenamiento con el agente ninja de 1000 partidas, los nuevos resultados obtenidos son los siguientes:

Agente	Partidas ganadas
Qlearning	45
Asesino	55

Tabla 7.6: Qlearning vs ninja - 100 partidas - ronda 2

Aunque se mejoran los resultados obtenidos en la prueba anterior, aún no es capaz de ganar más de la mitad de las partidas.

7.2. Comportamientos del agente

En resumen el agente Qlearning ha demostrado ser capaz de mejorar durante las pruebas realizadas, el resultado obtenido es óptimo aunque mejorable, se tendrá en cuenta todos los resultados obtenidos y conclusiones para una mejora en la siguiente versión. A continuación se detallan las virtudes y defectos de nuestro agente:

- **Virtudes:**
  - **Comportamientos inteligentes:** El agente es capaz de aprender comportamientos inteligentes a partir del refuerzo, entre ellos es capaz de esquivar una bala, interponerse en la bala por un aliado erido, etc.
  - **Aprendizaje en tiempo real:** Se ha demostrado que el agente evoluciona, adaptando los datos obtenidos en las partidas para mejorar. La matriz Q se va adaptando en función a la experiencia del agente.
- **Defectos:**
  - **Comportamiento en equipo:** Dado que se aplica Qlearning a nivel de unidad, no se han obtenido los resultados esperados de colaboración entre unidades y trabajo en equipo. Para mejorar este punto será necesario modificar la heurística.
  - **Error al asignar estado:** En las pruebas realizadas, se observa como en algunos casos no es capaz de diferenciar estados, o bien es asignado a un estado erróneo. Esto le lleva a cometer errores como recibir una bala, o lanzarse contra varios enemigos.

Capítulo 8

Planificación

En este capítulo se detalla la planificación llevada a cabo, este proceso se ha realizado al comienzo del proyecto, a continuación se detallarán las tareas más representativas necesarias para el desarrollo del software, además de sus dependencias y organización para la realización de cada una de ellas.

- **Definición de Objetivos:** El jefe de proyecto se encarga de la definición de objetivos, que consiste en analizar las necesidades del cliente y transformarlas en objetivos para su equipo.
- **Acuerdo de implementación:** Una vez se han asimilado los objetivos del proyecto el jefe de proyecto redacta el presupuesto para presentarlo al cliente, una vez ambas partes están de acuerdo se firma el presupuesto que da paso a la ejecución del proyecto.
- **Definición de requisitos:** El jefe de proyecto se encarga de supervisar la definición de requisitos, que se encargará de desarrollar el programador, asesorado del experto en IA, cuyo asesoramiento es esencial para que el sistema final dote de las herramientas necesarias para que los usuarios implementen sus agentes de forma óptima y dispongan de los recursos necesarios.
- **Definición de casos de uso:** El programador se encargará de definir los casos de uso a partir de los requisitos obtenidos en la tarea anterior.
- **Definición de la arquitectura:** La tarea de definir la arquitectura del sistema recae del programador, que basado en su experiencia en desarrollar sistemas similares deberá escoger la que mejor se adapte al proyecto y sea capaz de implementar después todas las funcionalidades en el periodo planificado.
- **Implementación del sistema:** Después de definir todas las tareas anteriores, el programador en base a los requisitos, casos de uso y arquitectura deberá desarrollar todo el sistema. A su vez debe estar en contacto con el responsable de calidad para que pueda desarrollar la batería de pruebas simultáneamente.

- **Diseño y ejecución de batería de pruebas:** El perfil encargado de realizar esta tarea será el responsable de calidad, que en colaboración con el programador tendrá que diseñar la batería de pruebas de forma paralela al desarrollo, para ello podrá hacer uso de los casos de uso del sistema, así como contactar con el jefe de proyecto en caso de requerir asesoramiento sobre el sistema final. Una vez diseñadas las pruebas y a la espera de que el programador finalice la tarea de implementación del sistema, el responsable de calidad se encarga de ejecutar la batería de pruebas y reportar todos las incidencias al programador para que sean solucionadas. Una vez completada esta tarea finaliza su intervención en el proyecto.

- **Implementación de agente IA en el sistema:** El experto en IA del equipo se encargará de desarrollar un agente de IA y probarlo en el sistema, deberá reportar si se han tenido en cuenta sus recomendaciones de asesoramiento y las incidencias ocasionadas en sus pruebas si es que las hay.

- **Cierre del proyecto:** El jefe de proyecto estará a cargo de asegurar que el software está finalizado con todos los objetivos y requisitos cumplidos. Una vez llegue a un acuerdo con el cliente la finalización del proyecto, se podrá dar por cerrado.

8.1. Planificación de tareas y diagrama Gantt

Una vez definidas las tareas el jefe de proyecto se encarga de realizar una estimación de horas para cada una de ellas y de las dependencias entre las mismas. A partir de las cuales se realiza el diagrama Gantt que se puede ver en las figuras 8.1 y 8.1. La estimación se ha realizado teniendo en cuenta la participación de los 4 perfiles anteriormente descritos y con una dedicación de 40 horas semanales trabajando de lunes a viernes.

	Nombre	Inicio	Fin
1	Definición de objetivos	01/09/2015	09/09/2015
2	Acuerdo de implementación	10/09/2015	14/09/2015
3	⊟**Definir requisitos**	**15/09/2015**	**28/09/2015**
4	Asesoramiento IA	15/09/2015	23/09/2015
5	Validación del programador	15/09/2015	28/09/2015
6	Definición casos de uso	29/09/2015	02/10/2015
7	Definición de la arquitectura	05/10/2015	15/10/2015
8	Implementación del sistema	16/10/2015	26/11/2015
9	Diseño de la batería de pruebas	20/11/2015	26/11/2015
10	Ejecución de batería de pruebas contra el sistema	27/11/2015	03/12/2015
11	Resolución de incidencias en pruebas	04/12/2015	17/12/2015
12	Implementación de agente IA en el sistema	27/11/2015	14/12/2015
13	Pruebas de Agente IA en el sistema	15/12/2015	18/12/2015
14	Cierre del proyecto	18/12/2015	24/12/2015

Figura 8.1: Tareas representativas

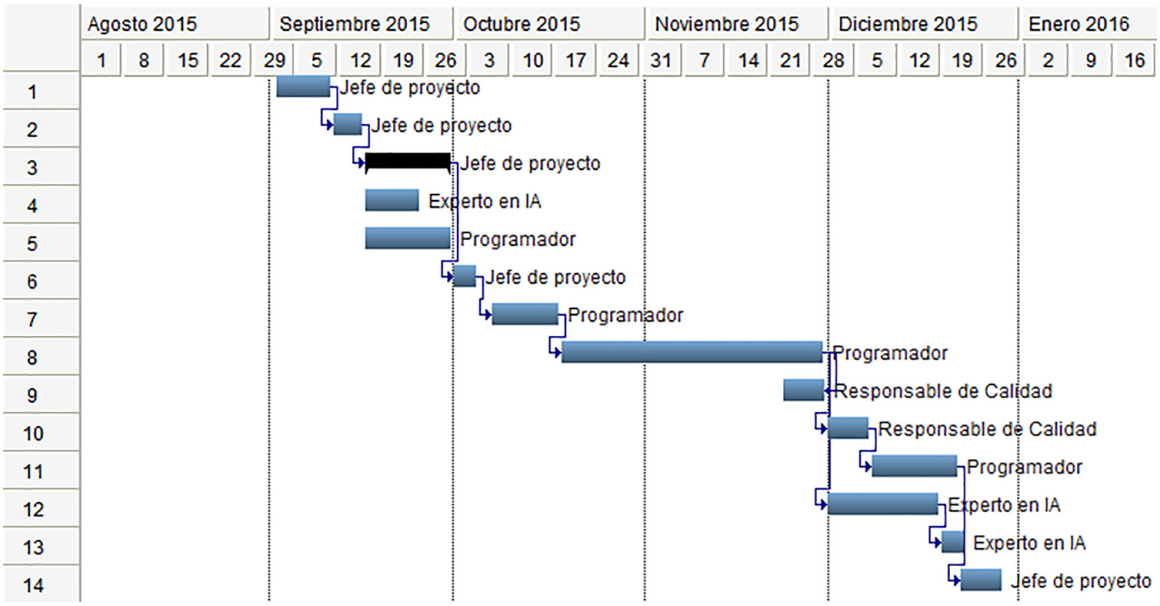

Figura 8.2: Diagrama Gantt planificación del proyecto

8.2. Metodologías ágiles

Dadas las características del equipo implicado en el proyecto, todos los miembros trabajan a distancia, el jefe de proyecto es el responsable de coordinar y organizar a todos los miembros del equipo. Haciendo de nexo del canal comunicativo entre todos los miembros. Para facilitar su tarea se decide la utilización de metodologías ágiles Scrum, el jefe de proyecto tomará el rol de Scrum Master, facilitando al resto del equipo la herramienta utilizada para el uso de esta metodología, en este caso para el proyecto la empresa provee de una herramienta de gestión de proyectos llamada Basecamp. El ciclo del proyecto estará marcado siguiendo las directrices de la metodología Scrum, de las cuales serán aplicadas las que más se correspondan con las necesidades del proyecto, en este caso el jefe de proyecto estará al cargo de un daily meeting de unos 15 minutos al comenzar la jornada, además de dividir el proyecto en Sprints de dos semanas, el proyecto consta de 4 meses de duración, por lo tanto estará dividido en 8 Sprints totales, en cada ciclo se reunirá todo el equipo en una videoconferencia de unos 45 minutos, determinando los objetivos del nuevo Sprint y analizando los resultados del Sprint anterior. La siguiente figura 8.2 muestra el flujo de trabajo llevado durante toda la duración del proyecto.

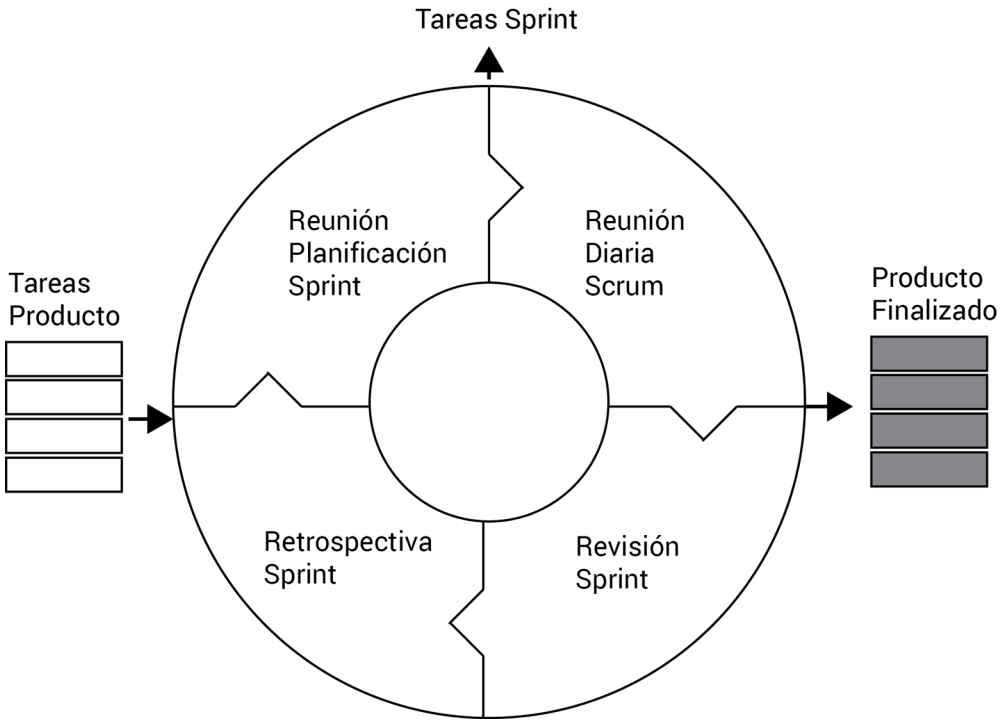

Figura 8.3: Diagrama de flujo de trabajo Scrum

Capítulo 9

Aspectos Económicos

En este capítulo se van a detallar tanto los aspectos económicos del proyecto como el análisis de los costes de desarrollo que serán necesarios para finalizar el sistema. Los costes están desglosados en recursos de personal, que implica todos los perfiles que están involucrados en el proyecto, coste de hardware, software y licencias, y por último los costes indirectos.

9.1. Perfiles requeridos

Los perfiles requeridos para realizar el proyecto son los siguientes:

- **Jefe de proyecto:** El jefe de proyecto es el encargado de definir los objetivos del proyecto, así cómo llegar a un acuerdo de implementación con el cliente. Además cuenta con la responsabilidad de organizar una planificación acorde con los requisitos. Es encargado de supervisar todo el proceso, garantizar la correcta ejecución de la planificación, y aportar soluciones a los problemas surgidos. A parte de servir de nexo comunicativo entre el resto de los miembros del equipo. El perfil recomendable para este puesto sería un ingeniero Informático con más de 3 años de experiencia en el sector.

- **Programador:** Este perfil estará encargado de diseñar los requisitos del software a partir de los objetivos, además será encargado de la implementación del sistema. Sin dejar de lado la arquitectura del sistema y su visión global del proyecto para facilitar el mantenimiento y mejoras del sistema en un futuro. Un perfil recomendable sería un graduado en informática con experiencia de al menos 2 años en sistemas similares.

- **Experto en IA:** Está encargado de asesorar sobre los requisitos necesarios del sistema para que la implementación de agentes con inteligencia artificial sea sencilla y la aplicación dote de todas las herramientas necesarias para que los usuarios puedan desarrollar sus agentes de la manera más óptima posible. Además se encargará de realizar un agente una vez finalizado el desarrollo, con el cuál se comprobará el correcto funcionamiento de la plataforma. El perfil recomendado para este

puesto sería un ingeniero informático que haya cursado la especialidad en computación, además de tener experiencia en competiciones de IA.

- **Responsable de calidad:** Estará encargado de asegurar la calidad del software que se ofrece, para ello tendrá que diseñar una batería de pruebas que aseguren el correcto funcionamiento del mismo. Posteriormente una vez el software esté en estado beta, deberá ejecutar dicha batería de pruebas y comunicar todas las anomalías encontradas al programador para que las solucione. Un perfil adecuado sería alguien con 2-3 años en el sector en un puesto similar.

Para cada uno de los perfiles, se les asignará una serie de tareas que tendrán que desarrollar. En la siguiente tabla 9.1 se muestran las tareas necesarias para el proyecto con el rol asociado y el tiempo en horas necesario para desarrollarlo.

Perfil	**Tarea**	**Horas**
Jefe de proyecto	Definición de objetivos	56
Jefe de proyecto	Acuerdo de implementación	24
Jefe de proyecto	Definir Requisitos	80
Experto en IA	Asesoramiento experto en IA	56
Programador	Validación de programador	80
Jefe de proyecto	Definición de casos de uso	32
Programador	Definición de la arquitectura	72
Programador	Implementación del sistema	240
Responsable de calidad	Diseño de la batería de pruebas	40
Responsable de calidad	Ejecución de batería de pruebas contra el sistema	40
Programador	Resolución de incidencias en pruebas	80
Experto en IA	Implementación de Agente IA en el sistema	96
Experto en IA	Pruebas de Agente IA en el sistema	32
Jefe de proyecto	Cierre del proyecto	40

Tabla 9.1: Tareas por perfil

9.2. Coste de personal

La contratación del personal requerido se ha realizado a través de una plataforma de búsqueda de freelance, se ha trabajado con anterioridad con todos los perfiles contactados, por lo tanto se sabe que trabajan bien y que son aptos para desarrollar el trabajo requerido.

Para realizar la estimación del coste bruto de los perfiles requeridos, se ha partido del coste por hora de los diferentes trabajadores y del total de horas computadas a las tareas que les atañe a cada uno de ellos, obteniendo como resultado el coste bruto total del coste de personal.

Perfil	**Tarea**	**Horas**	
Recurso	**Bruto/hora**	**Horas**	**Total bruto**
Jefe de proyecto	20 €	232	4640 €
Programador	8 €	472	3776 €
Experto en IA	12 €	184	2208 €
Responsable de calidad	8 €	80	640 €
Total			**11264 €**

Tabla 9.2: Coste bruto de personal

Al tratarse de trabajadores externos freelance no hay que pagar seguridad social, tal y cómo está obligado por ley para los empleados de una empresa. Por contrario el trabajador freelance emite una factura y por lo tanto hay que aplicarle al coste de horas que nos factura el IVA, que refleja el total que pagará la empresa por el servicio prestado, tal y como se observa en la tabla 9.3.

Recurso	**Total bruto**	**Total IVA**
Jefe de proyecto	4.640 €	5.614,4 €
Programador	3.776 €	4.568,96 €
Experto en IA	2.208 €	2.671,68 €
Responsable de calidad	640 €	774,4 €
Total	**11.264 €**	**13.629,44 €**

Tabla 9.3: Coste con IVA de personal

9.3. Costes hardware y software

Dado que todos los implicados en el proyecto son trabajadores freelance. En el contrato se especifica que cada uno es responsable de poner sus herramientas de trabajo, tanto componentes hardware, cómo el software requerido. Por lo tanto esa parte no hay que imputarla en gastos.

Sin embargo la empresa dota a los trabajadores del proyecto de varias herramientas online, para facilitar la comunicación y optimizar el trabajo realizado. Así como la fluidez a la hora de realizar las tareas. A continuación se detallan las herramientas necesarias y una explicación detallada del uso de las mismas.

- **Basecamp:** Es una herramienta de gestión de proyectos que permite realizar un seguimiento de las tareas que se están realizando además de servir de canal de comunicación entre los miembros del proyecto. La modalidad contratada, que añade funcionalidades a la básica, tiene un coste de 71.76€/mes.
- **Gitlab:** Se contrata un repositorio GIT con hosting al servicio del programador para gestionar las versiones del desarrollo, además de poder llevar un mejor registro de los cambios. Se ha escogido GitLab por que tiene las prestaciones que necesitamos para el proyecto. El coste asciende a 80€/mes.

El coste de dichas herramientas está detallado en la tabla 9.4, el cálculo de los servicios por subscripción mensual se realiza teniendo en cuenta el tiempo total de realización del proyecto, en este caso cuatro meses.

Software	**Coste**
Gitlab premium	320 €
basecamp	287,04 €
Total	**607,04 €**

Tabla 9.4: Coste herramientas

9.4. Costes indirectos

La gestión del personal se ha realizado a través de una agencia, que se encarga de encontrar los perfiles requeridos y tramitar los contratos con todo el personal freelance, esta agencia se queda con una comisión del diez porciento del total de coste de personal. Dichos gastos son imputados como gastos de gestión. Véase la tabla 9.5.

Otrosgastos	**Coste**
Gastos de gestión	1.362,94 €
Total	**1.362,94 €**

Tabla 9.5: Costes indirectos

Cómo los trabajadores freelance trabajan a distancia, para este proyecto no se imputan gastos de alquiler de oficina, tampoco de internet ni electricidad.

9.5. Resumen de costes

El total de los costes del proyecto, teniendo en cuenta los puntos anteriores, es el siguiente.

CONCEPTO	**Coste**
Recursos personas	13.629,44 €
Software	607,04 €
Costes indirectos	1.362,94 €
TOTAL	**15.599,42 €**

Tabla 9.6: Resumen costes brutos

9.6. Presupuesto

A la hora de realizar el presupuesto para el cliente, hay que tener en cuenta que al coste total calculado de **15599,42 €**, hay que sumarle los conceptos siguientes:

- **Margen de beneficio:** Para este proyecto, se ha asignado un margen de beneficios del 15 %, que aseguran que la empresa se lleve un porcentaje de beneficios del total del proyecto.

- **Riesgo e imprevistos:** Para costear los posibles imprevistos durante la realización del proyecto se añade un 8 % del total del proyecto.

Concepto	**Coste**
Total coste bruto	15.599,42 €
Riesgo e imprevistos	1.247,95 €
Margen de beneficios	2.339,91 €
Subtotal	19.187,29 €
Total con IVA	**23.216,62 €**

Tabla 9.7: Resumen costes presupuesto

Luis Sebastián Huerta
Calle melancolía, 4
28035 Madrid

Luis Sebastián Huerta, Calle melancolía, 4 , 28035 Madrid

Cliente
Nombre:
CIF:
Dirección:

Nº Factura	Email	Fecha
153	luis.sebastian@alumnos.uc3m.es	1. September 2015

Investigación y desarrollo de una máquina virtual que permita la ejecución de agentes inteligentes en JavaScript

Categoría	Cantidad	Descripción	Precio/hora	Total
Personal				
	232	Horas jefe de proyecto	29,77 €	6.905,71 €
	472	Horas programador	11,13 €	5.254,30 €
	184	Horas experto en IA	14,52 €	2.671,68 €
	80	Horas responsable de calidad	9,68 €	774,4 €
			Total personal:	13.629,44 €
Gastos de gestión				
	1	Gastos de gestión	5.557,85 €	5.557,85 €
			Total gestión:	5.557,85 €

Subtotal:	19.187,29 €
Impuestos (IVA 21%):	4.029,33 €
Total presupuesto:	**23.216,62 €**

Firma:
Luis Sebastián Huerta como jefe de proyecto

Validez del presupuesto:
Este presupuesto tiene una validez de 30 días naturales siempre y cuando esté sellado o firmado por uno de los responsables del proyecto, en caso contrario carecerá de

Capítulo 10

Conclusiones

En esta sección se van a analizar los aspectos positivos y negativos del proyecto, evaluar el grado de cumplimiento de los objetivos marcados inicialmente.

La versión final del proyecto cumple todos los objetivos y expectativas que tenía, estoy orgulloso de los resultados obtenidos, y sobre todo del apoyo recibido de compañeros y amigos. Su gran fascinación por nuestro proyecto me ha dado ánimos para conseguir terminarlo con una sonrisa, pero aún quedan algunos detalles que pulir y se seguirá mejorando la aplicación más adelante, una vez presentado el proyecto se va a liberar el código a la comunidad, para que en caso de no poder continuar nosotros con el proyecto, se pueda hacer cargo la comunidad.

Capítulo 11

Lineas futuras

11.1. Batallas para más de dos jugadores

Se plantea la opción de implementar en un futuro partidas para más de dos jugadores, ya sea por equipos o un todos contra todos.

El motor de juego esta preparado para controlar la ejecución de un número indeterminado de jugadores, pero serían necesarias modificaciones en los mapas adaptados al número de jugadores, en los cuales se indica la posición inicial de cada uno de los equipo.

11.2. Modificación del terreno del mapa

Para un futuro se plantea la implementación de diferentes tipos de terreno, la implementación es sencilla, habría que definir que propiedades tiene ese terreno y adaptar el comportamiento de las unidades en función a eso.

11.3. Diferentes tipos de unidades

Para añadir dinamismo a las partidas, se plantea crear diferentes tipos de unidades. Los atributos de las unidades son variables, y el sistema está preparado para generar nuevas unidades que hereden de esas, se pueden crear unidades más rápidas, o con proyectiles más rápidos.

Pero para ello será necesario adaptar los mapas para que indiquen la posición de las unidades nuevas.

11.4. Diferentes modos de juego

Como idea adicional, se podrían crear diferentes modos de juego cambiando el objetivo, por ejemplo en lugar de matar a todos los enemigos que se tenga que capturar la bandera enemiga.

Esto plantea muchos retos a diferencia de los anteriores, ya que para realizar esta implementación requiere modificar todo el comportamiento del juego, o crear uno nuevo.

11.5. Mejora de la máquina virtual

Se ha planteado la idea de en un futuro poder dar soporte a más lenguajes de programación para poder llegar a más programadores y que el desconocimiento del lenguaje implementado que es Javascript no sea una limitación. Dicha actualización no es difícil de implementar dado que el servidor trabaja mediante una salida de datos con el estado del juego(stdout), y una entrada de datos (stdin) con las acciones del agente.

Bibliografía

[1] Marcos Pérez Ferro. DISEÑO Y DESARROLLO DE UN CLIENTE Y UN SERVIDOR EN JAVASCRIPT PARA GESTIONAR BATALLAS Y CAMPEONATOS ENTRE AGENTES INTELIGENTES (JSWARS). (Spanish). *UC3M*, 2016.

[2] Waterloo Computer Science Club. Ai challenge. `http://research.microsoft.com/en-us/projects/trueskill/`.

[3] Microsoft Research. True skill ranking system. `http://research.microsoft.com/en-us/projects/trueskill/`.

[4] Peng Zhang Jochen Renz, XiaoYu (Gary) Ge and Stephen Gould. Aibirds. `http://aibirds.org/`.

[5] Julian Togelius Tom Schaul Diego Perez, Spyridon Samothrakis and Simon Lucas. General video game ai competition. `http://www.gvgai.net/`.

[6] Mario ai championship. `http://www.marioai.org/`.

[7] Mario ai championship. `http://www.marioai.org/`.

[8] Thorbjørn Lindeijer y 106 contribuidores en GitHub. Editor de mapas tiled. `http://www.mapeditor.org/`.